RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)

DR. BABASAHEB AMBEDKAR OPEN UNIVERSITY

AHMEDABAD

Editorial Panel

Author : Dr. Bhavik Pandya
Assistant Professor,
Navgujarat College of Computer Application,
Ahmedabad

&

Prof. Tejas B. Modi
Head & Associate Professor,
Kalol Institute of Management (MCA),
Kalol

Editor : Prof. (Dr.) Nilesh K. Modi
Professor & Director,
School of Computer Science,
Dr. Babasaheb Ambedkar Open University,
Ahmedabad

Language Editor : Jagdish Vinayakrao Anerao,
Associate Professor of English at
Smt AP Patel Arts & NP Patel Commerce
College Naroda, Ahmedabad.

ISBN 978-93-91071-03-5

Edition : 2021

Acknowledgment

Every attempt has been made to trace the copyright holders of material reproduced in this book. Should an infringement have occurred, we apologize for the same and will be pleased to make necessary correction/amendment in future edition of this book.

ROLE OF SELF–INSTRUCTIONAL MATERIAL IN DISTANCE LEARNING

The need to plan effective instruction is imperative for a successful distance teaching repertoire. This is due to the fact that the instructional designer, the tutor, the author (s) and the student are often separated by distance and may never meet in person. This is an increasingly common scenario in distance education instruction. As much as possible, teaching by distance should stimulate the student's intellectual involvement and contain all the necessary learning instructional activities that are capable of guiding the student through the course objectives. Therefore, the course / self–instructional material is completely equipped with everything that the syllabus prescribes.

To ensure effective instruction, a number of instructional design ideas are used and these help students to acquire knowledge, intellectual skills, motor skills and necessary attitudinal changes. In this respect, students' assessment and course evaluation are incorporated in the text.

The nature of instructional activities used in distance education self–instructional materials depends on the domain of learning that they reinforce in the text, that is, the cognitive, psychomotor and affective. These are further interpreted in the acquisition of knowledge, intellectual skills and motor skills. Students may be encouraged to gain, apply and communicate (orally or in writing) the knowledge acquired. Intellectual–skills objectives may be met by designing instructions that make use of students' prior knowledge and experiences in the discourse as the foundation on which newly acquired knowledge is built.

The provision of exercises in the form of assignments, projects and tutorial feedback is necessary. Instructional activities that teach motor skills need to be graphically demonstrated and the correct practices provided during tutorials. Instructional activities for inculcating change in attitude and behaviour should create interest and demonstrate need and benefits gained by adopting the required change. Information on the adoption and procedures for practice of new attitudes may then be introduced.

Teaching and learning at a distance eliminate interactive communication cues, such as pauses, intonation and gestures, associated with the face–to–face method of teaching. This is particularly so with the exclusive use of print media. Instructional activities built into the instructional repertoire provide this missing interaction between the student and the teacher. Therefore, the use of instructional activities to affect better distance teaching is not optional, but mandatory.

Our team of successful writers and authors has tried to reduce this.

Divide and to bring this Self–Instructional Material as the best teaching and communication tool. Instructional activities are varied in order to assess the different facets of the domains of learning.

Distance education teaching repertoire involves extensive use of self–instructional materials, be they print or otherwise. These materials are designed to achieve certain pre–determined learning outcomes, namely goals and objectives that are contained in an instructional plan. Since the teaching process is affected over a distance, there is need to ensure that students actively participate in their learning by performing specific tasks that help them to understand the relevant concepts. Therefore, a set of exercises is built into the teaching repertoire in order to link what students and tutors do in the framework of the course outline. These could be in the form of students' assignments, a research project or a science practical exercise. Examples of instructional activities in distance education are too numerous to list. Instructional activities, when used in this context, help to motivate students, guide and measure students' performance (continuous assessment)

PREFACE

We have put in lots of hard work to make this book as user–friendly as possible, but we have not sacrificed quality. Experts were involved in preparing the materials. However, concepts are explained in easy language for you. We have included many tables and examples for easy understanding.

We sincerely hope this book will help you in every way you expect.

All the best for your studies from our team!

RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)

Contents

Unit 15 **PRACTICAL SOLVING OF SQL**

Introduction, Practical Solving of SQL, Example 1 – Customer Contact Survey, Example 2 – SQL Creating SQL, Example 3 – SQL Creating Database Triggers, Example 4 – SQL Creating Command Language Scripts, Example 5 – MS–DOS Batch Command File, Example 6 – SQL* Plus Command File LIST6_16.SQL, Master/ Detail or Parent/Child SQL

Dr. Babasaheb Ambedkar Open University Ahmedabad **BCAR 302**

RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)

BLOCK 1 : BASIC CONCEPTS AND SQL* PLUS

BASIC CONCEPTS AND SQL* PLUS

Block Introduction :

This block provides an introduction to SQL* Plus. It provides an overview of how to run SQL* Plus and demonstrates this with various examples.

You can use the SQL*Plus program in conjunction with the SQL database language and its procedural language extension, PL/SQL. The SQL database language allows you to store and retrieve data in Oracle. PL/SQL allows you to link several SQL commands through procedural logic. SQL* Plus enables you to execute SQL commands and PL/SQL blocks, and to perform many additional tasks as well. The aim of this block is to give a basic introduction to the students of computers science about sql. This block is going to lay the foundation of SQL and database management system in the mind of the students. Every basic thing associated with SQL has been explained in this topic. The whole topic has been divided into two units.

First unit gives an introduction on SQL, it even discusses about the role of SQL in database management system. The unit further discusses about RDBMS and its various components. Second unit discusses further about the evolution of SQL* Plus and its various features. The students will also get a detailed insight a?out the various commands used in SQL along with the various reporting techniques. At last, the discussion has been made on SQL DBA which has even been discussed in very detail.

Block Objectives :

After learning this block, you will be able to understand :

- Database management system and Various types of key
- RDBMS and ORDBMS
- Database and Database model
- Evolution and SQL* Plus commands
- SQL DBA

Block Structure :

Unit 01

BASIC CONCEPTS

UNIT STRUCTURE

1.0 Learning Objectives :

After learning this unit, you will be able to understand :

- The theoretical and physical aspects of a relational database
- Oracle implementation of the RDBMS and ORDBMS
- New features of Oracle8I
- About Database and Database model
- The use and user of database

1.1 Introduction :

A Database is a collection of related information stored to provide the availability to many users for different purposes. The content of database is obtained by combining data from various sources in an organization. So that data are available to all users and redundant data can be minimized or eliminated. Database systems are designed to manage large bodies of information. The management of data involves both definition of structure for storage of information.

➢ **Database Management System :**

A database can be of any size and of varying complexity. For example a data consists of only a few hundred records, each with a simple structure. On the other hand, the card catalog of a large library may contain half a million cards stored under different categories – by primary author's last name, by subject, by book title–with each category organized in alphabetic order. A database of even greater size and complexity is maintained by the Internal Revenue Service to keep track of the tax forms filed by U.S. taxpayers. If we assume that there are 100 million taxpayers and if each taxpayer files an average of five form with approximately 200 characters of information per form, we would get a database of 10*(106)*200*5 characters (bytes) of information. If the IRS keeps the past three returns for each taxpayer in addition to the current return, we would get a database of 4*(10U) bytes (400 gigabytes). This huge amount of information must be organized and managed so that users can search for, retrieve, and update the data as needed. Creation of files, Addition of data, deletion of data, modification of data; Creation, addition and deletion of entire files.

- The data can be retrieved collectively and selectively.
- Stored data can be sorted or indexed according to Users direction and desecration.
- System can generate various reports. Thus report may be either standardized or generated according to user definition.
- Mathematical function can be performed and stored data can be manipulated with these functions to perform desired calculations. To maintain data integrity and Database use.

The DBMS process and interprets users request to retrieve information from database. These requests may be keyed directly from a terminal or coded as high level language programs to be submitted for either processing or interactive.

A database represents some aspect of the real world, sometimes called the mini–world or the Universe of Discourse (UOD). Changes to the mini–world is reflected in the database. A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a Database.

A database is designed, built, and populated with data for a Specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

➢ **Schemas, Instances, and Database State :**

In any data model it is important to distinguish between the description of the database and the database itself. The description of database is called the database schema, which is specified during database design and is not expected to change frequently Most data models have certain conventions for displaying the schemas as diagrams A displayed schema is called a schema diagram for the database shown in the diagram That displays the structure of each record type but not the actual instances of records. The data inside the database at a particular moment in time is called a database state or snapshot. It is also called the current set of occurrences or instances.

➢ **Relational Database Management System :**

A relationship is a logical linkage between two entities that describes how the entities are associated with each other. Creating a relationship explicitly defines an association between entities in the data model. RDBMS is an acronym for Relational Database Management System. The data in RDBMS is stored in database objects called tables. The database tables are the primary data storage for every RDBMS and essentially, they are collections of related data entries. For example, a table called Users might store information about many persons, and each entry in this table will represent one unique user. Even though all user entries in the Users table are unique, they are related in the sense that they describe similar objects. Each database table consists of columns and rows. Each table column defines the type of data stored in it, and this data type is valid for all rows in this table. A table row is a collection of data having 1 entry for each column in this particular table. RDBMS store the data into group of tables, which might or might not be related by common fields (database table columns).

➢ **Degree of Relationship :**

- One to One (1 : 1)
- One to Many (1 : N)
- Many to Many (M : N)

➢ **Attributes and Entities :**

This is second important data–modeling concept, which is to be understood. Attributes are additional characteristics or information defined for an entity. An entity's attributes do not define an entity, but they provide additional information about an entity' that may be useful elsewhere. Entities and Attributes enable to explicitly define what information is being stored in the database. The following should be kept in mind while implementing Entities and Attributes,

- Entities are modeled as tables.
- In a table, each instance of an entity is called a row.
- Attributes are modeled as columns in a table.
- Programmers often refer to rows and columns as records and fields.

Tables and columns are usually singular, this is relational modeling convention. Tables are almost always named after the entity they represent.

➢ **Primary key and Foreign Key :**

The primary key is special column in the table that can be used to identifying any one row. The important thing is that the value in the primary key column must be unique in the table, in which it resides. The primary key may be chosen out of the existing fields in a table, or a new field can be created expressly for this purpose. The easiest way to create a primary key is to create a special column called as identity column.

1.2 Data Base Management System :

Data is one of the most important assets of a company. It is very important to make sure that data is stored and maintained accurately and quickly. DBMS (Database Management System) is a system that is used to store and manage data.

A DBMS is a set of programs that is used to store and manipulation data.

Manipulation of data includes the following :

- Adding new data, for example adding details of new student.
- Deleting unwanted data, for example deleting the details of students who have completed course.
- Changing existing data, for example modifying the fee paid by the student.

A DBMS provides various functions like data security, data integrity, data sharing, data concurrence, data independence, data recovery etc. However, all data base management systems that are now available in the market like Sybase, Oracle, and MS–Access do not provide the same set of functions, though all are meant for data management.

Database managements systems like Oracle, DB2 are more powerful and meant for bigger companies. Whereas, database management systems like MS–Access are meant for small companies. So one has to choose the DBMS depending upon the requirement.

➢ **Features of DBMS :**

The following are main features offered by DBMS. Apart from these features different database management systems may offer different features. For instance, Oracle is increasing being fine–tuned to be the database for Internet applications. This may not be found in other database management systems. These are the general features of database management systems. Each DBMS has its own way of implementing it. ADBMS may have more features the features discussed here and may also enhance these features.

A. Support for Large Amount of Data :

Each DBMS is designed to support large amount of data. They provide special ways and means to store and manipulate large amount of data. Companies are trying to store more and more amount of data. Some of this data will have to be online (available every time).

In most of the cases the amount of data that can be stored is not actually constrained by DBSM and instead constrained by the availability of the hardware. For example, Oracle can store terabytes of data.

B. Data Sharing, Concurrency and Locking :

DBMS also allows data to be shared by two or more users. The same data can be accessed by multiple users at the same time – which is known as data concurrency. However when same data is being manipulated at the same time by multiple users certain problems arise. To avoid these problems, DBMS locks data that is being manipulated to avoid two users from modifying the same data at the same time. The locking mechanism is transparent and automatic. Neither we have to inform to DBMS about locking nor need we to know how and when DBMS is locking the data. However, as a programmer, if we can know intricacies of locking mechanism used by DBMS, we will be better programmers.

C. Data Security :

While DBMS allowing data to be shared, it also ensures that data in only accessed by authorized users. DBMS provides features needed to implement security at the enterprise level. By default, the data of a user cannot be accessed by other users unless the owner gives explicit permissions to other users to do so.

D. Data Integrity :

Maintaining integrity of the data is an import process. If data loses integrity, it becomes unusable and garbage. DBMS provides means to implement rules to maintain integrity of the data. Once we specify which rules are to be implemented, then DBMS can make sure that these rules are implemented always. Three integrity rules – domain, entity and referential are always supported by DBMS.

E. Fault Tolerance and Recovery :

DBMS provides great deal of fault tolerance. They continue to run in spite of errors, if possible, allowing users to rectify the mistake in the meantime. DBSM also allows recovery in the event of failure. For instance, if data on the disk is completely lost due to disk failure, then also data can be recovered to the point of failure if proper back up of the data is available.

F. Support for Languages :

DBMS supports a data access and manipulation language. The most widely used data access language for RDBMS (relational database management systems) is SQL. DBMS implementations of SQL will be compliant with SQL standards set by ANSI. Apart from supporting a non–procedural language like SQL to access and manipulate data DBMS now a day also provides a procedural language for data processing. Oracle supports PL/SQL and SQL Server provides T–SQL.

G. Entity and Attribute :

An entity is any object that is stored in the database. Each entity is associated with a collection of attributes. For example, if you take a data of a training institute, student is an entity as we store information about each student in the database. Each student is associated with certain values such as roll number, name and course etc., which are called as attributes of the entity.

■ **RDBMS Definitions :**

➢ **Database Management Largely Involves :**

- Storage of Data
- Manipulation of the data
- Access restriction for unauthorized users.

➢ **Functions of a DBMS :**

Some of the functions of DBMS are :

- Database Definition – how data is to be stored and organized.
- Database Creation – Storing data in a defined database
- Data Retrieval – Querying and reporting
- Updating – changing the contents of the database.
- Programming user facilities for system development.
- Database revision and restructuring
- Database Integrity control
- Performance monitoring.

➢ **Characteristics of DBMS :**

- Control of data redundancy
- Sharing data
- Maintenance of integrity
- Support for transaction control and recovery

- Data independence
- Availability of productivity tools
- Security
- Processing speeds
- Hardware independence

❑ **Check Your Progress – 1 :**

1. ________ is one of the most important assets of a company.

   a. Data   b. Database

2. A ________ is a set of programs that is used to store and manipulation data.

   a. DBMS   b. data

3. Database managements systems like Oracle, DB2 are more powerful and meant for ________ companies.

   a. Bigger   b. small

4. DBMS also allows data to be ________ by two or more users.

   a. shared   b. transferred

5. DBMS provides ________ deal of fault tolerance.

   a. Great   b. low

1.3 Various Types of Keys :

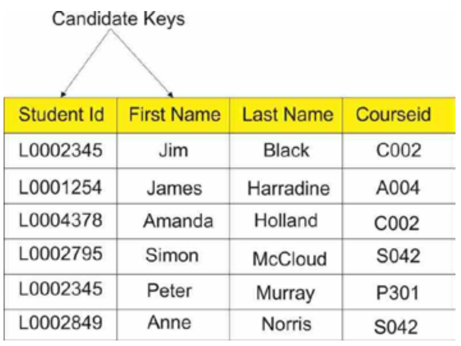
Keys are the key part of a relational database and a vital part of the structure of a table. They ensure each record within a table can be uniquely identified by one or a combination of fields within the table. They help enforce integrity and help identify the relationship between tables. There are three main types of keys, candidate keys, primary keys and foreign keys. There is also an alternative key or secondary key that can be used, as the name suggests, as a secondary or alternative key to the primary key.

➢ **Super Key :**

A Super key is any combination of fields within a table that uniquely identifies each record within that table.

➢ **Candidate Key :**

A candidate is a subset of a super key. A candidate key is a single field or the least combination of fields that uniquely identifies each record in the table. The least combination of fields distinguishes a candidate key from a super key. Every table must have at least one candidate key but at the same time can have several.


Candidate Keys

Student Id	First Name	Last Name	Courseid
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0004378	Amanda	Holland	C002
L0002795	Simon	McCloud	S042
L0002345	Peter	Murray	P301
L0002849	Anne	Norris	S042

Fig 1.1 Candidate Keys

As an example we might have a student_id that uniquely identifies the students in a student table. This would be a candidate key. But in the same table we might have the student's first name and last name that also, when combined, uniquely identify the student in a student table. These would both be candidate keys.

In order to be eligible for a candidate key it must pass certain criteria.

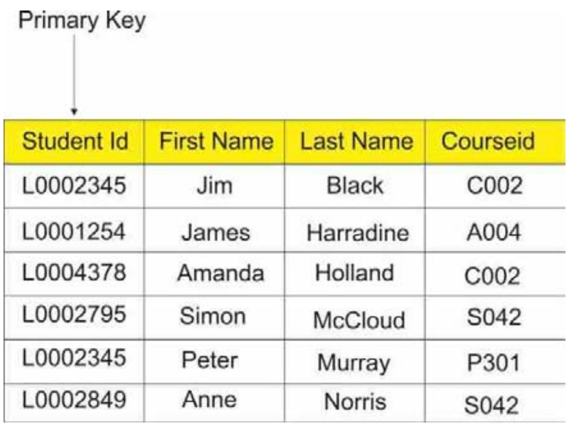
- It must contain unique values
- It must not contain null values
- It contains the minimum number of fields to ensure uniqueness
- It must uniquely identify each record in the table

Once your candidate keys have been identified you can now select one to be your primary key.

➢ **Primary Key :**

A primary key is a candidate key that is most appropriate to be the main reference key for the table. As its name suggests, it is the primary key of reference for the table and is used throughout the database to help establish relationships with other tables. As with any candidate key the primary key must contain unique values, must never be null and uniquely identify each record in the table.

As an example, a student id might be a primary key in a student table, a department code in a table of all departments in an organisation. This module has the code DH3D 35 that is no doubt used in a database somewhere to identify RDBMS as a unit in a table of modules. In the table below we have selected the candidate key student_id to be our most appropriate primary key.


Primary Key

Student Id	First Name	Last Name	Courseid
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0004378	Amanda	Holland	C002
L0002795	Simon	McCloud	S042
L0002345	Peter	Murray	P301
L0002849	Anne	Norris	S042

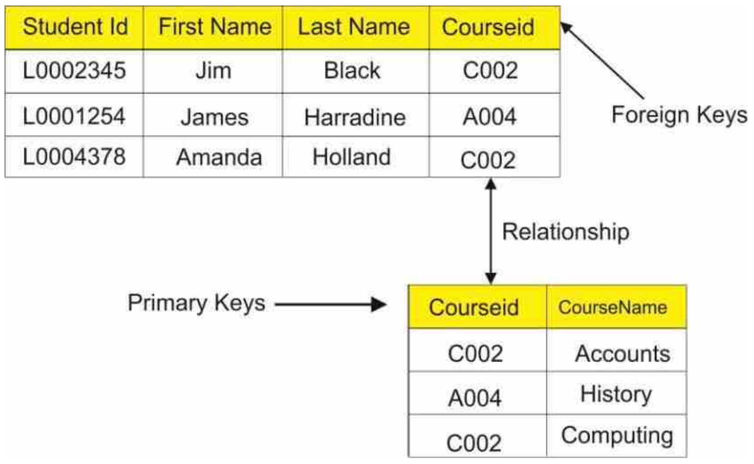
Fig 1.2 Primary Key

Primary keys are mandatory for every table each record must have a value for its primary key. When choosing a primary key from the pool of candidate keys always choose a single simple key over a composite key.

➢ **Foreign Key :**

A foreign key is generally a primary key from one table that appears as a field in another where the first table has a relationship to the second. In other words, if we had a table A with a primary key X that linked to a table B where X was a field in B, then X would be a foreign key in B.

An example might be a student table that contains the course_id the student is attending. Another table lists the courses on offer with course_id being the primary key. The 2 tables are linked through course_id and as such course_id would be a foreign key in the student table.



Student Id	First Name	Last Name	Courseid
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0004378	Amanda	Holland	C002

Courseid	CourseName
C002	Accounts
A004	History
C002	Computing

Fig 1.3 Foreign Key

➢ **Secondary Key or Alternative Key :**

A table may have one or more choices for the primary key. Collectively these are known as candidate keys as discuss earlier. One is selected as the primary key. Those not selected are known as secondary keys or alternative keys.

For example, in the table showing candidate keys above we identified two candidate keys, student Id and first Name + last Name. The student Id would be the most appropriate for a primary key leaving the other candidate key as secondary or alternative key. It should be noted for the other key to be candidate keys, we are assuming you will never have a person with the same first and last name combination. As this is unlikely, we might consider fist Name + last Name to be a suspect candidate key as it would be restrictive of the data you might enter. It would seem a shame to not allow John Smith onto a course just because there was already another John Smith.

➢ **Simple Key :**

Any of the keys described before (i.e., primary, secondary or foreign) may comprise one or more fields, for example if first Name and last Name was our key this would be a key of two fields where as studentId is only one. A simple key consists of a single field to uniquely identify a record. In addition, the field in itself cannot be broken down into other fields, for example, studentId, which uniquely identifies a particular student, is a single field and therefore is a simple key. No two students would have the same student number.

➢ **Compound Key :**

A compound key consists of more than one field to uniquely identify a record. A compound key is distinguished from a composite key because each field, which makes up the primary key, is also a simple key in its own right. An example might be a table that represents the modules a student is attending. This table has a studentId and a module Code as its primary key. Each of the fields that make up the primary key are simple keys because each represents a unique reference when identifying a student in one instance and a module in the other.

➢ **Composite Key :**

A composite key consists of more than one field to uniquely identify a record. This differs from a compound key in that one or more of the attributes, which make up the key, are not simple keys in their own right. Taking the example from compound key, imagine we identified a student by their first Name + last Name. In our table representing students on modules our primary key would now be first Name + last Name + module Code. Because firstName + lastName represent a unique reference to a student, they are not each simple keys, they have to be combined in order to uniquely identify the student. Therefore, the key for this table is a composite key.

❑ **Check Your Progress – 2 :**

1. ________ are, as their name suggests, a key part of a relational database and a vital part of the structure of a table.

   a. Keys b. lock

2. A ________ is any combination of fields within a table that uniquely identifies each record within that table.

   a. Super key b. Supreme key

3. A candidate is a ________ of a super key. A candidate key is a single field or the least combination of fields that uniquely identifies each record in the table.

   a. Subset b. Superset

4. A ________ key is a candidate key that is most appropriate to be the main reference key for the table.

   a. Primary b. Secondary

5. A ________ key is generally a primary key from one table that appears as a field in another where the first table has a relationship to the second.

   a. Foreign b. Primary

1.4 Normalization and Relational Algebra :

■ **Normalization :**

➢ **Functional dependency :**

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2, ..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (→) that is, X → Y, where X functionally determines Y. The left–hand side attributes determine the values of attributes on the right–hand side.

➢ **Axioms Armstrong's :**

If F is a set of functional dependencies then the closure of F, denoted as F+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules that, when applied repeatedly, generates a closure of functional dependencies.

• **Reflexive rule –** If alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.

- **Augmentation rule** – If a $\rightarrow$ b holds and y is attribute set, then ay $\rightarrow$ by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- **Transitivity rule** – Same as transitive rule in algebra, if a $\rightarrow$ b holds and b $\rightarrow$ c holds, then a $\rightarrow$ c also holds. a $\rightarrow$ b is called as a functionally that determines b.

➢ **Trivial Functional Dependency :**

- **Trivial** – If a functional dependency (FD) X $\rightarrow$ Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.
- **Non–trivial** – If an FD X $\rightarrow$ Y holds, where Y is not a subset of X, then it is called a non–trivial FD.
- **Completely non–trivial** – If an FD X $\rightarrow$ Y holds, where x intersect Y = Φ, it is said to be a completely non–trivial FD.

➢ **Normalization :**

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **Update anomalies :** If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
- **Deletion anomalies :** We tried to delete a record, but parts of it were left undeleted because of unawareness and the data is also saved somewhere else.
- **Insert anomalies :** We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

➢ **First Normal Form :**

First Normal Form is defined in the definition of relations (tables) itself.

This rule defines that all the attributes in a relation must have atomic domains.

The values in an atomic domain are indivisible units.

Table First Normal Form

Course	Content
Programming	Java, c++
Web	Html, PHP, ASP

We re–arrange the relation (table) as below, to convert it to First Normal Form.

Table First Normal Form

Course	Content
Programming	Java
Programming	C++
Web	HTML
Web	PHP
Web	ASP

Each attribute must contain only a single value from its predefined domain.

➢ **Second Normal Form :**

Before we learn about the second normal form, we need to understand the following :

- **Prime attribute** – An attribute, which is a part of the prime–key, is known as a prime attribute.
- **Non–prime attribute** – An attribute, which is not a part of the prime–key, is said to be a non–prime attribute.

If we follow second normal form, then every non–prime attribute should be fully functionally dependent on prime key attribute. That is, if $X \rightarrow A$ holds, then there should not be any proper subset Y of X for which $Y \rightarrow A$ also holds true.

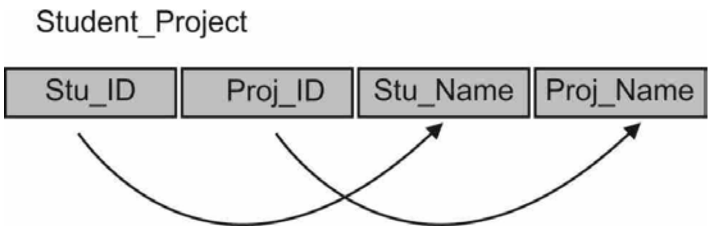


Fig 1.4 Relation not in 2NF

We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non–key attributes, i.e., Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called partial dependency, which is not allowed in Second Normal Form.

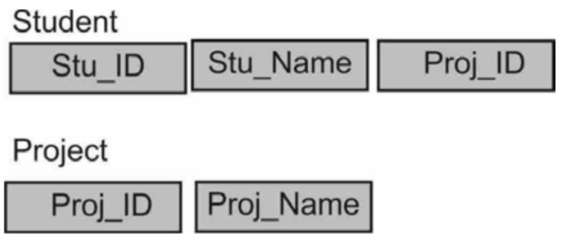


Fig 1.5 Relations in 2NF

We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

➢ **Third Normal :**

For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy:

- No non–prime attribute is transitively dependent on prime key attribute.
- For any non–trivial functional dependency, X → A, then either:
  - X is a superkey or,
  - A is prime attribute.

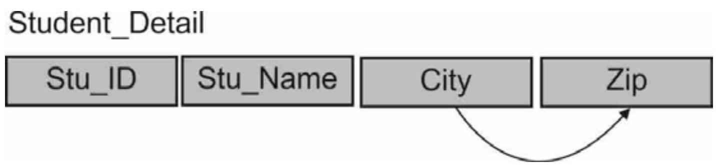


Fig 1.6 Relations not in 3NF

We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip neither is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists transitive dependency.

To bring this relation into third normal form, we break the relation into two relations as follows :

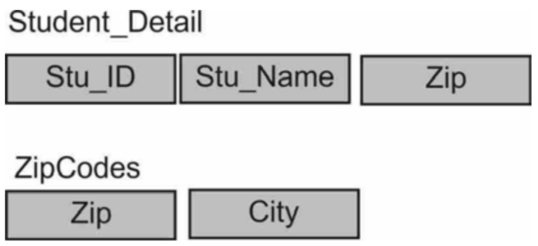


Fig 1.7 Relations in 3NF

➢ **Boyce–Codd Normal Form :**

Boyce–Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that –

- For any non–trivial functional dependency, X → A,
- X must be a super–key.

In the above image, Stu_ID is the super–key in the relation Student_Detail and Zip is the super–key in the relation ZipCodes. So,

Stu_ID → Stu_Name, Zip

and

Zip → City

Which confirms that both the relations are in BCNF?

➢ **Relational Algebra :**

Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages: relational algebra and relational calculus.

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary.They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows :

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

➢ **Select Operation (σ)**

It selects tuples that satisfy the given predicate from a relation.

Notation : σp(r)

Where σ stands for selection predicate and r stands for relation.p is prepositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like: =, ≠, ≥, <, >, ≤.

For example : σsubject=―database‖(Books)

Output : Selects tuples from books where subject is ‗database'.

σsubject=―database‖ and price=―450‖(Books)

Output : Selects tuples from books where subject is ‗database' and ‗price' is 450.

σsubject=―database‖ and price < ―450‖ or year > ―2010‖(Books)

Output : Selects tuples from books where subject is ‗database' and ‗price' is 450 or those books Published after 2010.

➢ **Project Operation (Π)**

It projects column(s) that satisfy a given predicate.

Notation : ΠA1, A2, An (r)

Where A1, A2, An are attribute names of relation r.

Duplicate rows are automatically eliminated, as relation is a set.

For example : Πsubject, author (Books)

Selects and projects columns named as subject and author from the relation Books.

➢ **Union Operation (∪)**

It performs binary union between two given relations and is defined as :

$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$

Notion : $r \cup s$

Where r and s are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold :

- Rand s must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

Π author (Books) ∪ Π author (Articles)

Output : Projects the names of the authors who have either written a book or an article or both.

➢ **Set Difference (–)**

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation : r – s

Finds all the tuples that are present in r but not in s.

Πauthor (Books) – Πauthor (Articles)

Output : Provides the name of authors who have written books but not articles.

Cartesian Product (X) Cartesian Product (X)

Combines information of two different relations into one.

Notation : r X s

Where r and s are relations and their output will be defined as :

r X s = { q t | q ∈ r and t ∈ s}

Πauthor = ‗tutorialspoint‘ (Books X Articles)

Output – Yields a relation, which shows all the books and articles written by tutorialspoint

Rename Operation – The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. ‗rename‘ operation is denoted with small Greek letter rho ρ.

Notation – ρ x (E)

Where, the result of expression E is saved with name of x.

Additional operations are:

- Set intersection
- Assignment

❑ **Check Your Progress – 3 :**

1. ________ is a set of constraints between two attributes in a relation.

   a. Functional dependency b. database

2. Functional dependency is represented by ________ sign.

   a. An Arrow b. A Bow

3. If F is a set of functional dependencies then the closure of F, denoted as ________.

   a. F+ b. F++

4. ________ are a set of rules that, when applied repeatedly, generates a closure of functional dependencies.

   a. Axioms b. Arrows

5. If a ________ design is not perfect, it may contain anomalies.

   a. Database b. Data structure

1.5 Components of RDBMS :

A relational database is an electronic structure for storing information. It consists primarily of tables of related information. The tables include rows and columns, also known as records and variables. The relational databases are defined by schema, which indicate the relationship among the tables. The tables include primary keys that identify specific records within a single table and foreign keys that are used to connect tables with each other.

➢ **Schema :**

A database schema explains the structure and relationship of tables and other elements within a single relational database. The most common type of schema is a logical schema, which displays those relationships, as well as specific attributes of variables within the related tables.

➢ **Tables :**

Tables are composed of rows and columns, or records and variables. The records contain specific information about a unique characteristic. Variables display the type of information contained within each row. An individual record, for example, might include information about a customer named —James Jones.|| The corresponding variable for the customer might be —Name.||

➢ **Keys :**

Primary keys are variables that are used to identify unique records within a table. Each record in a relational database table should have a unique primary key composed of one or more variables. Foreign keys are variables used to join related tables.

❑ **Check Your Progress – 4 :**

1. A ________ is an electronic structure for storing information.

   a. relational database b. database

2. The ________ include rows and columns, also known as records and variables.

   a. Tables b. Data

3. The relational databases are defined by________, which indicate the relationship among the tables.

   a. Schema b. Table

4. A ________ explains the structure and relationship of tables and other elements within a single relational database.

   a. database schema b. table

5. ________ are composed of rows and columns, or records and variables.

a. Tables b. schema

1.6 Let Us Sum Up :

This unit gives unique learning on Structured Query Language and it helps to make practice on SQL commands which provides immediate results. SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc. SQL is an ANSI (American National Standards Institute) standard but there are many different versions of the SQL language.

Data is one of the most important assets of a company. It is very important to make sure data is stored and maintained accurately and quickly. On the other hand a DBMS provides various functions like data security, data integrity, data sharing, data concurrence, data independence, data recovery etc. Database managements systems like Oracle, DB2 are more powerful and meant for bigger companies.

Keys are, as their name suggests, a key part of a relational database and a vital part of the structure of a table. They ensure each record within a table can be uniquely identified by one or a combination of fields within the table. They help enforce integrity and help identify the relationship between tables.

A relational database is an electronic structure for storing information. It consists primarily of tables of related information. The relational databases are defined by schema, which indicate the relationship among the tables.

1.7 Answer for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a	**2 :** a	**3 :** a	**4 :** a	**5 :** a

❑ **Check Your Progress 2 :**

1 : a	**2 :** a	**3 :** a	**4 :** a	**5 :** a

❑ **Check Your Progress 3 :**

1 : a	**2 :** a	**3 :** a	**4 :** a	**5 :** a

❑ **Check Your Progress 4 :**

1 : a	**2 :** a	**3 :** a	**4 :** a	**5 :** a

1.8 Glossary :

1. **Keys :** They are, as their name suggests, a key part of a relational database and a vital part of the structure of a table.
2. **DBMS :** It is a set of programs that is used to store and manipulation data.

1.9 Assignment :

Explain DBMS and its features.

1.10 Activities :

Write a note on various keys on RDBMS.

1.11 Case Study :

Explain the benefits of studying Relational algebra.

1.12 Further Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt. Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw–Hill Publication.
3. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
4. Database Management System by Raghu Ramkrishnan – Tata McGraw–Hill Publication.
5. SQL, PL/SQL : The Programming Language Oracle – Ivan Bayross – BPB Publication.

Unit 02

INTRODUCTION TO ORACLE TOOLS /SQL* PLUS

UNIT STRUCTURE

2.0 Learning Objectives :

After learning this block, you will be able to understand :

- Evolution of SQL* Plus.
- Various SQL* Plus commands
- About SQL DBA
- Command reference under SQL DBA

2.1 Introduction :

SQL* Plus is the command–line interface to the Oracle database. Its fundamental reason for existence is to allow you to enter and execute ad hoc SQL statements and PL/SQL code blocks. This unit explains what SQL* Plus is, how it relates to other Oracle tools (as well as the database), and why you should master it. At the end of the unit I'll introduce you to the sample data, which is used for many of the examples in this book. If you like, you can load that data into your database and test out each example as you go through this book.

SQL* Plus is essentially an interactive query tool, with some scripting capabilities. It is a non–GUI, character–based tool that has been around since the dawn of the Oracle age. Using SQL* Plus, you can enter an SQL statement, such as a SELECT query, and view the results. You can also execute Data Definition Language (DDL) commands that allow you to maintain and modify your database. You can even enter and execute PL/SQL code. In spite of SQL* Plus's age and lack of —flash,|| it is a workhorse tool used day in and day out by database administrators, developers, and yes, even end users.

2.2 Evolution of SQL* Plus :

SQL* Plus has been around for a long time, pretty much since the beginning of Oracle. In fact, the original author was Bruce Scott. Any DBA will recognize the name Scott. It lives on, immortalized as the owner of a set of example tables that used to be installed with every version of Oracle and that you still can install even today. The original purpose of SQL* Plus can be summed up in the succinct words of Kirk Bradley, another early author of SQL* Plus, who told me, —We needed a way to enter statements into the database and get results.||

Kirk‘s reason is still arguably the major reason most people use SQL* Plus today, more than 15 years after it was originally written. SQL* Plus certainly satisfies a compelling and enduring need.

The original name of the product was not SQL* Plus. The original name was UFI, which stands for User Friendly Interface. This name has its roots in one of the first relational database systems ever developed, IBM‘s System R, the product of a research effort by IBM. Some of IBM's documents referred to the command–line interface as the UFI, and that name was adopted by Oracle for its interactive SQL utility.

One of the more interesting uses of Oracle for UFI was as a tool to produce its documentation. The SQL* Plus DOCUMENT command, now considered obsolete, was used for this purpose. Script files were created that contained the manual text, interspersed with the SQL statements needed for the examples. The DOCUMENT command was used to set off the manual text so it would just be copied to the output file. When these scripts were run, the text was copied, the SQL statements were executed, and the result was documentation complete with examples.

SQL* Plus maintains a fascinating relic from the old days in the form of the SET TRIMOUT, SET TRIMSPOOL, and SET TAB commands. These commands control the printing of trailing spaces and the use of tabs to format columnar output. To understand why these commands even exist, you have to realize that when SQL* Plus first made its appearance, people thought that a dialup speed of 1200 bits per second (bps) was fast. In those days you could get your results much faster by avoiding the need to transmit large numbers of space characters across a dialup connection.

If you had a lot of whitespaces in your report, you spent a lot of time watching spaces print across your screen. In that environment, trimming spaces and using tabs to format columns provided a huge gain in throughput. Today, with our 10–megabit–per–second (10 Mbps) LAN connections and our 56–KB modems, we hardly give this a thought.

During the mid–1980s, Oracle experimented with efforts to add procedural capabilities to UFI. The result of this effort was AUFI, which stood for Advanced User Friendly Interface. AUFI implemented such things as IF statements and looping constructs, and was demonstrated publicly at an International Oracle User Group meeting in 1986 by Ken Jacobs, who is now Oracle's Vice President, Product Strategy, Oracle Server Technologies.

In spite of the public demos, whether or not to release AUFI as a shipping product was the subject of some debate within Oracle. Layering a procedural language on top of the existing UFI command set was proving difficult. It was made more difficult by the need to maintain full, backward compatibility so existing scripts written by Oracle's clients would not suddenly break when those clients upgraded. Because of these issues, the code to support the procedural enhancements became complex and somewhat unreliable. The issues of reliability and complexity led to Oracle's ultimate decision to kill the product, so AUFI never shipped. With the later advent of PL/SQL, procedural logic was supported within the database, and efforts to support a procedural scripting language were then seen as unnecessary. The name AUFI lives on in the name of the temporary file created when you use the SQL* Plus EDIT command. That file is named afiedt.buf. Even today, AFI is the prefix used for all the source code.

With the release of Oracle 5.0 in 1985, the name of the interactive query utility was changed from UFI to SQL* Plus. Most changes since then have been evolutionary. Each new release brings with it a few new commands and new options on existing commands. Some commands have been made obsolete, but many of these obsolete commands are still supported for purposes of backward compatibility.

Only a couple of truly significant changes to SQL* Plus have occurred over the years. In 1988, with the release of Oracle8i Database, Server Manager's STARTUP, SHUTDOWN, and RECOVER commands were implemented in SQL* Plus, which was designated the primary, command–line interface into the Oracle database. Server Manager was deprecated, and by the time Oracle 9i Database came along, Server Manager no longer existed.

By far the most significant and visible change to SQL* Plus in recent years has been the introduction of iSQL* Plus, a three–tier application that gives you access to SQL* Plus functionality via any standard we? browser.

As a prelude to iSQL* Plus, it was first necessary to give SQL* Plus the ability to generate output in HTML form. This took place in Release 8.1.7, which introduced the SET MARKUP command.

iSQL* Plus was first released in 2001 as a Windows–only application (part of Oracle9i Database Release

1. iSQL* Plus was expanded to most other operating systems in 2002 with the release of Oracle9i Database Release
2. Beginning with the release of Oracle Database 10g, iSQL* Plus is supported across all platforms.

2.2.1 Significant Features of SQL* Plus :

Some of the following significant features of SQL* Plus make it possible for you to set up command files that allow end–user input:

1. Defining user Variables
2. Substituting Values in Commands
3. Using the START Command to Provide Values
4. Prompting for Values

1. Defining user Variables :

You can define variables, called user variables, for repeated use in a single command file by using the SQL* Plus command DEFINE. Note that you can also define user variables to use in titles and to save you key strokes (by defining a long string as the value for a variable with a short name).

To define a user variable EMPLOYEE and give it the value —SMITH||, enter the following command :

```
SQL> DEFINE EMPLOYEE = SMITH
```

To confirm the definition of the variable, enter DEFINE followed by the variable name:

```
SQL> DEFINE EMPLOYEE
```

SQL* Plus lists the definition :

```
DEFINE EMPLOYEE = —SMITH|| (CHAR)
```

To list all user variable definitions, enter DEFINE by itself at the command prompt. Note that any user variable you define explicitly through DEFINE takes only CHAR values (that is, the value you assignto the variable is always treated as a CHAR datatype). You can define a user variable of datatype NUMBER implicitly through the ACCEPT command. To delete a user variable, use the SQL* Plus command UNDEFINE followed by the variable name.

2. Substituting Values in Commands :

If you want to write a query like the one in SALES to list the employees with various jobs, not just those whose job is SALESMAN. You could do that by editing a different CHAR value into the WHERE clause each time you run the command, but there is an easier way. By using a substitution variable in place of the value SALESMAN in the WHERE clause, you can get the same results you would get if you had written the values into the command itself. A substitution variable is a user variable name preceded by one or two ampersands (&). When SQL* Plus encounters a substitution variable in a command, SQL* Plus executes the command as though it contained the value of the substitution variable, rather than the variable itself.

For example, if the variable SORTCOL has the value JOB and the variable

MYTABLE has the value EMP, SQL* Plus executes the commands:

```
SQL> BREAK ON & SORTCOL
SQL> SELECT & SORTCOL, SAL
2 FROM & MYTABLE
3 ORDER BY & SORTCOL;
```

as if they were

```
SQL> BREAK ON JOB
SQL> SELECT JOB, SAL
2 FROM EMP
3 ORDER BY JOB;
```

The BREAK command suppresses duplicate values of the column named in SORTCOL.

3. Using the START Command to Provide Values :

You can use substitution variables anywhere in SQL and SQL* Plus commands, except as the first word entered at the command prompt. When SQL* Plus encounters an undefined substitution variable in a command, SQL*

Plus prompts you for the value. You can enter any string at the prompt, even one containing blanks and punctuation. If the SQL command containing the reference should have quote marks around the variable and you do not include them there, the user must include the quotes when prompted.

SQL* Plus reads your response from the keyboard, even if you have redirected terminal input or output to a file. If a terminal is not available (if, for example, you run the command file in batch mode), SQL* Plus uses the redirected file.

After you enter a value at the prompt, SQL* Plus lists the line containing the substitution variable twice: once before substituting the value you enter and once after substitution. You can suppress this listing by setting the SET command variable VERIFY to OFF.

Create a command file named STATS, to be used to calculate a subgroup statistic (the maximum value) on a numeric column :

```
SQL> CLEAR BUFFER
SQL> INPUT
SELECT &GROUP_COL,
MAX(&NUMBER_COL) MAXIMUM
FROM &TABLE
GROUP BY &GROUP_COL
SQL> SAVE STATS
Created file STATS
```

Avoiding Unnecessary

Prompts for Values

3–20 SQL* Plus User's Guide and Reference

Now run the commands file STATS and respond as shown below to the prompts for values:

```
SQL> @STATS
Enter value for group_col: JOB
old 1: SELECT &GROUP_COL,
new 1: SELECT JOB,
Enter value for number_col: SAL
old 2: MAX(&NUMBER_COL) MAXIMUM
new 2: MAX(SAL) MAXIMUM
Enter value for table: EMP
old 3: FROM &TABLE
new 3: FROM EMP
Enter value for group_col: JOB
old 4: GROUP BY &GROUP_COL
new 4: GROUP BY JOB
```

SQL* Plus displays the following output :

```
JOB          MAXIMUM
ANALYST      3000
CLERK        1300
MANAGER      2975
PRESIDENT    5000
SALESMAN     1600
```

If you wish to append characters immediately after a substitution variable, use a period to separate the variable from the character. For example :

```
SQL> SELECT * FROM EMP WHERE EMPNO=‗&X.01‘;
Enter value for X: 123will be interpreted as
SQL> SELECT * FROM EMP WHERE EMPNO=‗12301‘;
```

4. Prompting for Values :

Suppose you wanted to expand the file STATS to include the minimum, sum, and average of the ―number‖ column. You may have noticed that SQL* Plus prompted you twice for the value of GROUP_COL and once for the value of NUMBER_COL in Example 3–12, and that each GROUP_COL or NUMBER_COL had a single ampersand in front of it. If you were to add three more functions–using a single ampersand before each—to the command file, SQL* Plus would prompt you a total of four times for the value of the number column.

You can avoid being re–prompted for the group and number columns by adding a second ampersand in front of each GROUP_COL and NUMBER_COL in STATS. SQL* Plus automatically DEFINEs any substitution variable preceded by two ampersands; but does not DEFINE those preceded by only one ampersand. When you have DEFINED a variable, SQL* Plus substitutes the value of variable for each substitution variable referencing variable (in the form and variable or &variable). SQL* Plus will not prompt you for the value of variable in this session until you UNDEFINE variable.

2.2.2 SQL* Plus Commands :

SQL* Plus is a command–line tool that provides access to the Oracle RDBMS. SQL* Plus which makes you to :

- Enter SQL* Plus commands to configure the SQL* Plus environment
- Startup and shutdown an Oracle database
- Connect to an Oracle database
- Enter and execute SQL commands and PL/SQL blocks
- Format and print query results

SQL* Plus is available on several platforms. In addition, it has a web–based user interface, iSQL* Plus.

The commands shown in Table A–1 are SQL* Plus commands available in the command–line interface. Not all commands or command parameters are shown.

Table A–1 Basic SQL* Plus Commands

How To...	SQL* Plus Command
Log in to SQL* Plus	SQLPLUS [{ username[/password][@connect_identifier] | / } [AS { SYSDBA | SYSOPER }] | /NOLOG]
List help topics available in SQL* Plus	HELP [INDEX | topic]
Execute host commands	HOST [command]
Show SQL* Plus system variables or environment settings	SHOW { ALL | ERRORS | USER | system_variable | ... }
Alter SQL* Plus system variables or environment settings	SET system_variablevalue
Start up a database	STARTUP PFILE = filename [MOUNT [dbname] | NOMOUNT | ...]
Connect to a database List column definitions for a table, view, or synonym, or specifications for a function or procedure	CONNECT [[username [/password] [@connect_identifier] [/ AS { SYSOPER | SYSDBA }]] DESCRIBE [schema.] object
Edit contents of the SQL buffer or a file	EDIT [filename [.ext]]
Get a file and load its contents into the SQL buffer	GET filename [.ext] [LIST | NOLLIST]
Save contents of the SQL buffer to a file	SAVE filename [.ext] [CREATE | REPLACE | APPEND]
List contents of the SQL buffer	LIST [n | nm | n LAST | ...]
Delete contents of the SQL buffer	DEL [n | nm | n LAST | ...]
Add new lines following current line in the SQL buffer	INPUT [text]
Append text to end of current line in the SQL buffer	APPEND text

Find and replace first occurrence of a text string in current line of the SQL buffer	CHANGE sepcharold [sepchar [new [sepchar]]] sepchar can be any non–alphanumeric character such as —/|| or —!||
Capture query results in a file and, optionally, send contents of file to default printer	SPOOL [filename [.ext] [CREATE | REPLACE | APPEND | OFF | OUT]
Run SQL* Plus statements stored in a file	@ { url | filename [.ext] } [arg...] START filename [.ext] [arg...] .ext can be omitted if the filename extension is .sql
Execute commands stored in the SQL buffer	/
List and execute commands stored in the SQL buffer	RUN
Execute a single PL/SQL statement or run a stored procedure	EXECUTE statement
Disconnect from a database	DISCONNECT
Shut down a database	SHUTDOWN [ABORT | IMMEDIATE | NORMAL | ...]
Log out of SQL* Plus	{ EXIT | QUIT } [SUCCESS | FAILURE | WARNING | ...] [COMMIT | ROLLBACK]

❑ **Check Your Progress – 1 :**

1. The original author of SQL was ________.

   a. Bruce Scott   b. Kirk Bradley

2. The original purpose of SQL* Plus can be summed up in the succinct words of ________.

   a. Kirk Bradley   b. Bruce Scott

3. You can use ________ variables anywhere in SQL and SQL* Plus commands.

   a. Substitution   b. Complimentary

4. You can define variables, called ________, for repeated use in a single command file by using the SQL* Plus command DEFINE.

   a. user variables   b. substitution

2.3 Reporting Techniques :

A report is information provided in a neat and understandable format. Report contains details as well as summary information. We have so far seen different clauses of SELECT command. But no clause allows you to display details as well as summary information. For instance, GROUP BY can be used to get summary information but it cannot display details. Simple SELECT can display details but cannot display summary information.

SQL* PLUS is a tool that is used to send SQL command to Oracle Instance. It is an environment in which you enter SQL commands. It also provides some extra commands such as DESCRIBE, EDIT etc., which are called as SQL* PLUS commands. SQL* PLUS environment also provides a set of commands which can be used to generate report.

The following is a sample report that we are going to generate in this unit. This report is generated with SQL* PLUS commands that are specifically meant for this purpose and SELECT command, which is used to retrieve data from database.

➢ **Report Script :**

The following is the report script. The script is used to generate the report shown above. The following sections will explain the commands used in the script.

Type the script in any text editor under the name like payreport.sql and then run it from SQL prompt of SQL* Plus as follows.

```
SQL>>start c:\orabook\payreport.sql
rem ***********************************************
rem Purpose : Script to generate Payments Report
rem AUthor : P.Srikanth
rem Date : 10–Oct–2001
rem Place : Visakhapatanam
rem ***********************************************
rem set break and compute settings
break on report on course skip page on batch skip 2
compute sum of amount on batch course report
set pagesize 24
set linesize 90
set feedback off
column amount format 99,999
column name format a20 heading ‗Student
Name‘ column dj heading ‗Date of|Joining‘
column dp heading ‗Date of|Payment‘
title skip 1 right ‗Page:‘ format 99 sql.pno skip 1 center ‗Payments
Report‘ skip 2
spool payreport.lst
select c.ccode course, b.bcode batch, p.rollno, s.name name, phone, dj,
dp, amount
from batches b, students s, payments p, courses c
whereb.ccode = c.ccode and b.bcode = s.bcode and s.rollno = p.rollno
order by course, batch;
```

spool off

set feedback on

rem clear settings

clear compute

clear break

clear column

ttitle off

➢ **BREAK Command :**

Specifies how which column(s) the data selected by SELECT command is to be grouped (broken). This is also used to specify what should be done when break is given.

BRE[AK] [ON expression [action]] . . .

Expression is the column on which the data is to be grouped or ROW or REPORT keyword.

Action specifies what action should be taken when break is issued. The following are the possible actions.

SKI[P] n Prints specified number of empty lines.

SKI[P] page Issues a page break at break.

DUP[LICATE] Prints the value of a break column in every selected row.

NODUP[LICATE] Prints blanks instead of the value of a break column when the value

isa duplicate of the column's value in the preceding row.

Note : BREAK command alone displays the current break settings

The following BREAK command issues break whenever it detects a change in BCODE column

and starts a new page.

break on bcode skip page

Note : It is important to give ORDER BY clause on the columns that are used in BREAK command.

When multiple columns are used in BREAK, SQL* PLUS issues break starting from right most column to left most.

break on country skip page on city skip 2

First it issues break on CITY and then issues break on COUNTRY. You need to makes sure the order is right.

BREAK command in script

The script above used the following break.

break on report on course skip page on batch skip 2

The above BREAK issues break on three different levels. First whenever there is a change in BATCH column, second whenever there is a change in COURSE column and finally at the end of the report.

It is important to know the order in which columns are to be given – most specific to most common.

For this break to work properly the ORDER BY clause of the SELECT must be given as follows :

Order by course, batch

➢ **COMPUTE Command :**

This is used to compute and print summary information. The

COMP[UTE] [function [LABEL text] OF {column}...

ON {column | REPORT | ROW} . . .]

FUNCTION is any of the functions listed in Table 1.

If you give more than one function, use spaces to separate the functions.

Table List of functions used in COMPUTE

FUNCTION	COMPUTES	COMPUTES
AVG	Average of non–null values	NUMBER
COU[NT]	Count of non–null values	all types
MAX[IMUM]	Maximum value	NUMBER, CHAR, VARCHAR, VARCHAR2
MIN[IMUM]	Minimum value	NUMBER, CHAR, VARCHAR, VARCHAR2
NUM[BER]	Count of rows	all types
STD	Standard deviation of non–null values	NUMBER
SUM	Sum of non–null values	NUMBER
VAR[IANCE]	Sum of non–null values	NUMBER

Note : Every COMPUTE requires a corresponding BREAK. COMPUTE displays summary information, only when there is abreak on the given level. The following COMPUTE will display the su?total and grand total of amount.

Compute sum of amount on bcode report

For the above COMPUTE to function there must be a corresponding BREAK as follows.

break on ?code skip page on report

LABEL keyword specifies the text to be displayed as label for the computed value. The maximum length of the label is 500 characters.

compute sum label ?Grand Total' of amount on report

NOTE : The label is truncated to the size of first column in the SELECT

➢ **COMPUTE Command in Script :**

The following compute command is used in the script to generate the report.

compute sum of amount on batch course report

It will display the sum of AMOUNT column at the end of each batch, course and at the end of report.

➢ **COLUMN Command :**

Specifies the display attributes of a column. The following are the possible options :

Text for the column heading

Alignment of the column heading

Format of NUMBER type columns

Wrapping of text

COL[UMN] [{column|expr} [option ...]]

The following is the list of a few commonly used options of COLUMN command.

FOR[MAT] format

HEA[DING] text

JUS[TIFY] {L[EFT]|C[ENTER]|C[ENTRE]|R[IGHT]}

NUL[L] char

WRA[PPED]|WOR[D_WRAPPED]|TRU[NCATED]

➢ **HEADING :**

Specifies the heading to be displayed. If no heading is given, the name of the column is displayed as heading.

If heading separator (|) is used then heading is split into multiple lines.

JUSTIFY is used to align heading either to LEFT, CENTER or RIGHT. column ccode heading ‗Course|Code' justify center NULL

Substitutes null values with the given value in the display. If not given then null is displayed as blank.

COLUMN command may also be used to show display attributes of a particular column as follows :

SQL> column name

COLUMN name ON

HEADING ‗Student | Name ‗ headsep ‗|'

JUSTIFY center

COLUMN command in script

The following COLUMN commands in the script are used to format columns.

Column amount format 99,999

column name format a20 heading ‗Student

Name' column dj heading ‗Date of|Joining'

column dp heading ‗Date of|Payment'

AMOUNT column is formatted as 99,999.

NAME column is displayed with a width of 20 columns and heading is set to ―Student Name‖.

DJ's heading is set to two lines heading where first line is ―Date of‖ and second line is ―Joining‖. The same is the case with DP column.

TTITLE and BTITLE Commands

TTITLE specifies the text to be displayed at the top of each printed page.

BTITILE displays

textat the bottom of the printed page.

TTI[TLE] [options [text]] [ON | OFF]

BTI[TLE] [options [text]] [ON | OFF]

The following is the list of options:

Option Meaning

BOLD Prints text in bold print.

COL n Prints title after n columns in the current line.

ENTER Aligns title to center.

LEFT Aligns title to left.

RIGHT Aligns title to right.

SKIP n Skips n number of lines before printing title.

TTITLE center ‗First line ‗skip 1 center ‗Second line‘

Prints the text First Line in the first line and after skipping one line then displays text Second Line.

TTITLE left ‗Payments Report' right sql.pno

Prints the text Payments Report on the left and current page number on the right.

SQL.PNO

returns the current page number. The other variables that you can use are :

SQL.USER –

current username, SQL.RELEASE – current Oracle release number.

TTITLE off

Will turn off display of top title.

TTITLE command in script

The following TTITLE command is used to display the top title for payments report.

ttitle skip 1 right ‗Page:‘ format 99 sql.pno skip 1 center ‗Payments

Report' skip 2

Oracle For Beginners Page : 11

srikanthtechnologies.com

First string ―Page:‖ is displayed followed by the page number of the current page. These two are right justified to the page. Then one line is skipped and then at the center of the line title

―Payments Report‖ is displayed. Then two lines are skipped before the data of the page is displayed.

➢ **SPOOL Command :**

Stores the result of query into an operating system file. It can be used to send the content of the file to printer.

Once spooling is turned on whatever that is displayed on the screen, a copy of that is copied

into spool file.

SPO[OL] [filename | OUT | OFF]

Filename Specifies the name of the file into which output is to be stored. It begins spooling.

OFF Stops spooling.

OUT Stops spooling and sends the contents of the spool file to default printer.

The following SPOOL command sends subsequent output to file

REPORT.LST. SPOOL report.lst

SPOOL command in the script

SPOOL command is used in the script to capture the output into file

PAYREPORT.LST as

follows.

spool payreport.lst

Then after the SELECT command SPOOL command is used again to stop capturing the output

using :

spool off

Oracle For Beginners Page : 12

srikanthtechnologies.com

CLEAR Command

Resets the current value of the specified option.

CLEAR option

Where option may be any of the following.

Option what it clears?

BRE[AKS] Break settings set by BREAK.

COMP[UTES] Compute setting set by COMPUTE.

SCR[EEN] Content of SQL* PLUS window.

COL[UMNS] Column setting set by COLUMN.

BUFF[ER] Text in the current buffer.

The following CLEAR command will rest all summary information specified by COMPUTE.

■ **Clear Computes :**

➢ **SET Variables :**

Set variables/system variables are used to configure SQL* PLUS environment. For instance, system variables can be used to set page pause, the length of page, size of line and so on.

SET command is used to change the value of system variables.

SET system_variable value

The following is a list of a few system variables. For the complete list of system variables and all the possible options, please see on–line help.

Some of the system variables can store more than one value. For instance, FEEDBACK stores a number and ON/OFF state.

System variable Meaning

AUTOCOMMIT If set to ON, automatically commits changes made by SQL command.

FEEDBACK If set to ON, displays the number of rows selected by SELECT, if number of rows is >= the number specified by FEEDBACK variable.

LINESIZE Specifies the number of characters that can be displayed in a singleline.

PAGESIZE Specifies the number of lines to be displayed in a single page.

NUMFORMAT Specifies the default numeric format for numbers.

PAUSE If set to ON, gives pause at the end of each page. Also contains the text to be displayed when pause is issued.

SERVEROUTPUT If set to ON, enables the display of output by DBMS_OUTPUT package.

TERMOUT If set to OFF, suppresses the display of output of start file.

Table 3 : SET Variables.

The following SET commands will configure SQL* Plus for report.

set pagesize 50

set linesize 132

set feedback off

set pause off

Displaying Information Using SHOW Command

You can display the current values of system variables and other options using SHOW command.

SHO[W] option

Where option may be any of the following.

Option Meaning

System_variable Any system variable.

ERR[ORS] Compilation error of procedures, functions.

REL[EASE] The current release of Oracle.

TTI[TLE] and BTI[TLE] Current setting of title.

USER The name of the current user.

ALL Displays the values of all show options.

For the complete list of options, please see on–line help.

To get the name of the current user, enter :

```
SQL> show user
user is –SRIKANTH||
```

To display the current value of PAGESIZE variable, enter :

```
SQL> show pagesize
pagesize 24
```

❑ **Check Your Progress – 2 :**

1. A ________ is information provided in a neat and understandable format.

   a. Report b. Command

2. SQL* PLUS is a tool that is used to send SQL command to ________ Instance.

   a. Oracle b. Command

3. ________ Command specifies how which column(s) the data selected by SELECT command is to be grouped (broken).

   a. BREAK b. Stop

2.4 Let Us Sum Up :

In this unit, we have learnt that Although SQL* Plus has been around for a long time, pretty much since the beginning of Oracle. It lives on, immortalized as the owner of a set of example tables that used to be installed with every version of Oracle and that you still can install even today. The original purpose of SQL* Plus can be summed up in the following words.? We even learnt some of the significant features of SQL* Plus are defining user variables, substituting values in commands, using the START command to provide values and prompting for values

A report is information provided in a neat and understandable format. Report contains details as well as summary information. In the above discussion even the various SQL* DBA commands have been discussed in very detail.

2.5 Answer for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a **2 :** a **3 :** a 4 : a

❑ **Check Your Progress 2 :**

1 : a **2 :** a **3 :** a

2.6 Glossary :

1. **SQL* Plus :** It is the command–line interface to the Oracle database.
2. **Report :** It is information provided in a neat and understandable format.

2.7 Assignment :

Throw some light on the evolution of SQL * Plus took place.

2.8 Activities :

Discuss in detail the various significant features of SQL* Plus.

2.9 Case Study :

Discuss the role of reporting techniques.

2.10 Furthers Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw–Hill Publication.
3. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
4. Database Management System by Raghu Ramkrishnan – Tata McGraw–Hill Publication.
5. SQL, PL/SQL : The Programming Language Oracle – Ivan Bayross – BPB Publication.

Unit 03

SQL DBA

UNIT STRUCTURE

3.0 Learning Objectives :

After learning this block, you will be able to understand :

- About SQL DBA
- Command reference under SQL DBA

3.1 Introduction to SQL DBA :

Every organization using a database management system (DBMS) to manage data requires a database administration group to ensure the effective use and deployment of the company's databases. Since most modern organizations of any size use a DBMS, the need for a database administrator (DBA) is greater today than ever before. However, the discipline of database administration is neither well understood nor universally practiced in a coherent and easily replicated manner.

A DBA is the information technician responsi?le for ensuring the ongoing operational functionality and efficiency of an organization's databases and the applications that access those databases. The long answer to that question requires a book to answer–this book. This text will define the management discipline of database administration and provide practical guidelines for the proper implementation of the DBA function.

➢ **Why learn DBA ?**

Data is at the center of today's applications; today's organizations simply cannot operate without data. In many ways, business today is data. Without data, businesses would not have the ability to manage finances, conduct transactions, or contact their customers. Databases are created to store and organize this data. The better the design and utility of the database, the better the organization will be positioned to compete for business.

Indeed, one of the largest problems faced by IT organizations is ensuring quality database administration. A survey of IT managers conducted by Information

Week in December 2000 showed that the top two database management execution issues faced by companies are ease of administration and availability of qualified administrators.

Both of these issues were cited by 58% of survey respondents. Additionally, the 1999 Market Compensation Survey conducted by people3, a Gartner Company, shows that DBA positions take longer to fill than any other position. Clearly, there is no lack of demand for DBA skills in today's job market.

➢ **Data Administration :**

Data administration separates the business aspects of data resource management from the technology used to manage data; it is more closely aligned with the actual business users of data. The data administrator (DA) is responsible for understanding the business lexicon and translating it into a logical data model.

Referring back to the ADLC, the DA would be involved more in the requirements gathering, analysis, and design phase, the DBA in the design, development, testing, and operational phases.

Another difference between a DA and a DBA is the focus of effort. The DA is responsible for the following tasks:

- Identifying and cataloguing the data required by business users.
- Producing conceptual and logical data models to accurately depict the relationship among data elements for business processes.
- Creating an enterprise data model that incorporates all of the data used by all of the organization's business processes.
- Setting data policies for the organization.
- Identifying data owners and stewards.
- Setting standards for control and usage of data.

In short, the DA can be thought of as the Chief Data Officer of the corporation. However, in my experience, the DA is never given an executive position, which is unfortunate. Many IT organizations state that they treat data as a corporate as set, a statement that is believed by their actions. Responsibility for data policy is often relegated to technicians who fail to concentrate on the non–technical business aspects of data management. Technicians do a good job of ensuring availability, performance, and recoverability, but are not usually capable of ensuring data quality and setting corporate policies.

In fact, data is rarely treated as a true corporate asset. Think about the assets that every company has in common: capital, human resources, facilities, and materials. Each of these assets is modelled: charts of account, organization charts, reporting hierarchies, building blueprints, office layouts, and bills of material. Each is tracked and protected. Professional auditors are employed to ensure that no discrepancies exist in a company's accounting of its assets. Can we say the same thing about data?

A mature DA organization is responsible for planning and guiding the data usage requirements throughout the organization. This role encompasses how data is documented, shared, and implemented company wide. A large responsibility of the DA staff is to ensure that data elements are documented properly, usually in a data dictionary or repository. This is another key differentiation between a DA and a DBA. The DA focuses on the repository, whereas the DBA focuses on the physical databases and DBMS.

Furthermore, the DA deals with metadata, as opposed to the DBA, who deals with data. Metadata is often described as data about data; more accurately, metadata is the description of the data and data interfaces required by the business.

Data administration is responsible for the business's metadata strategy. Examples of metadata include the definition of a data element, business names for a data element, any abbreviations used for that element, and the data type and length of the element. Data without metadata is difficult to use. For example, the number 12 is data, but what kind of data? In other words, what does that 12 mean?

Without metadata, we have no idea. Consider this : Is the number 12

- A date representing December, the twelfth month of the year?
- A date representing the twelfth day of some month?
- An age?
- A shoe size?
- Or, heaven forbid, an IQ?

And so on. However, there are other, more technical aspects of metadata, too.

Think about the number 12 again.

- Is 12 a large number or a small one?
- What is its domain (that is, what is the universe of possible values of Which 12 is but a single value)?
- What is its data type? Is it an integer or a decimal number with a 0 scale?

Metadata provides the context by which data can be understood and therefore become information. In many organizations, metadata is not methodically captured and catalogued; instead, it exists mostly in the minds of the business users. Where it has been captured in systems; it is spread throughout multiple programs in file definitions, documentation in various states of accuracy, or in long lost program specifications. Some of it, of course, is in the system catalog of the DBMS. A comprehensive metadata strategy enables an organization to understand the information assets under its control and to measure the value of those assets.

One of the biggest contributions of data administration to the corporate data asset is the creation of data models. A conceptual data model outlines data requirement at a very high level. A logical data model provides in–depth details of data types, lengths, relationships, and cardinality. The DA uses normalization techniques to deliver sound data models that accurately depict the data requirements of an organization.

Many DBAs dismiss data administration as mere data modelling, required only because someone needs to talk to the end users to get the databases requirements. However, a true DA function is much more than mere data modelling. It is a business–oriented management discipline responsible for the data asset of the organization.

- Planning Database Creation
    - Plan the tables and indexes – no of tables, indexes, normalization, data–types, constraints, block size, location and storage parameters
    - Layout of the operating system – location of redolog files, data files, specifying the number of rows in data–block, storage requirements
    - Using oracle managed files

  Operating system files managed by oracle, manage disk space allocation
    - Determining Global database name
        - To be used by all users within network
    - Considering parameter file
        - Oracle provides parameter file init.ora that contains various initialization parameters
    - Selecting Database character set
    - Specifying the time zone
    - Specifying block size
    - Backup strategy
    - Operating the database

  When you create a database, Oracle create two user accounts by default:
- **SYS :**

  The SYS user account is automatically assigned the DBA role.

  When the database is created, Oracle creates data dictionary, tables, and views for the database in the SYS schema. You cannot manipulate the tables and views in the SYS schema.
- **SYSTEM :**

  The SYSTEM user account is automatically assigned the DBA role. The default password for using the SYSTEM account is MANAGER.

  Oracle stores the additional tables and views, which stores the information for administrative task in the SYSTEM schema. The database users should not be allowed to create tables and views in the SYSTEM schema.
- Oracle creates a role called DBA when database is created. The DBA role has all administrative privileges.
- This role should be granted to a single DBA who is responsible for administering and managing the database.
- **DBA Privileges**
    - An administrator requires specific privileges to perform database operations. Oracle provides two system privileges for administrators:
        - SYSDBA
        - SYSOPER
- Managing Profiles and Users

  Profile : It specifies system resources that are available to a database user while working with a database.

- System resources include CPU time available for processing in user session, logical IO operations, simultaneous sessions that can be established with the oracle server.
- You can specify a default profile that is used when no profile is assigned to the user.

The Database Administration consists of managing three important files namely Control Files, Data Files and Redo–Log files

➢ **Control Files :**

- A control file stores information about the physical structure of a database.
- It includes information about all the files within the database.
- A control file stores the content in binary format.
- Every database should have one control file in any case.
- Control file initialization parameter is defined in init.ora
- Control files are generated automatically when database is created.
- If control file name is not specified then default name is used
- Default is creation of control files during database creation
- By default only one control file is created
- More than two control files should be created because these files are needed in recovery operations
- It restores database in consistent state in case of failure
- If we are using a single control file and it gets corrupted, recovery is not possible
- Multiple copies of control files must be created on several drives

➢ **Data Files :**

- Data files are those files that store logical structures of the database.
- The data files are stores in a tablespace
- After assigning the data files to a tablespace you cannot remove it but can resize them.
- A tablespace is logical division of a database which contains many datafiles.
- Each database has atleast one tablespace (default is SYSTEM)
- Each tablespace can be associated with only one database

➢ **Online Redo Log Files :**

- Oracle records all the changes in the database in redo log files.
- Each database must have atleast two redo log files
- Redo Log files contains Redo Records
- Redo records contains set of change vectors
- Change vector describes the changes made to a single block in database
- Oracle reads the change vectors and automatically applies those changes in the pertinent blocks

- Transaction occurs in database
- Transactions are written to redo log buffers
- Log Writer (LGWR) writes the contents of the redo log buffers to online redo log files in batches.
- LGWR writes the redo records from log buffer to the first file in database.
- Then a System Change Number (SCN) is assigned to the redo records to recognize them. When all records are written on redo log files then transaction is declared as committed.
- When log files are filled completely LGWR returns to first redo log file and overwrites it with new data.
- If database is running in ARCHIVELOG mode, the database duplicates the online redo log files before overwriting them.
- This files would later used to recover the data.

❑ **Check Your Progress – 1 :**

1. The discipline of ________ is neither well understood nor universally practiced in a coherent and easily replicated manner.

   a. database administration b. DBMS

2. A ________ is the information technician responsible for ensuring the ongoing operational functionality and efficiency of an organization's databases and the applications that access those databases.

   a. DBA b. DBMS

3. ________ is at the center of today's applications.

   a. Data b. DBMS

4. Data is rarely treated as a true corporate ________

   a. Asset b. Liability

5. A mature ________ organization is responsible for planning and guiding the data usage requirements throughout the organization.

   a. DA b. DBMS

3.3 SQL* DBA Command Reference :

Command	**Function**
ALTER DATABASE	Changes storage group or log for database
ALTER DBAREA	Changes the size of a database area.
ALTER EXTERNAL FUNCTION	Changes an external function definition
ALTER PASSWORD	Changes a password.
ALTER STOGROUP	Adds or drops a database area from a storage group.
ALTER TABLE	Modifies the definition of a tale.
ALTER TABLE (error messages)	Makes error messages specific to a particular referential integrity violation.

ALTER TABLE (referential integrity)	Adds or drops primary and foreign keys.
ALTER TRIGGER	Enables and disables triggers defined on tables.
AUDIT MESSAGE	Writes a message string to an audit file.
CHECK DATABASE	Checks database for integrity.
CHECK INDEX	Checks specified index for integrity.
CHECK TABLE	Checks specified table for integrity.
COMMENT ON.	Replaces or adds a comment to the description of a table, view, column, or external function in the system catalog
COMMIT	Ends a logical unit of work and commits database changes made by it.
CREATE DATABASE	Physically creates a database.
CREATE DBAREA	Creates a database area.
CREATE EXTERNAL FUNCTION	Creates an external function.
CREATE INDEX	Creates an index on a table.
CREATE STOGROUP	Creates a storage group.
CREATE SYNONYM Defines	An alternate name for a table, view, or external function.
CREATE TABLE	Defines a table.
CREATE TRIGGER	Creates a trigger.
CREATE VIEW	Defines a view of one or more tables or views.
DBATTRIBUTE	Sets database–specific attributes
DEINSTALL DATABASE	Takes a database off the network, making it unavailable to users.
DELETE	Deletes one or more rows from a table.
DROP DATABASE	Physically deletes a database.
DROP DBAREA	Physically deletes a database area.
DROP EXTERNAL FUNCTION	Deletes an external function.
DROP INDEX	Removes an index.
DROP STOGROUP	Deletes a storage group.
DROP SYNONYM	Deletes a synonym
DROP TABLE	Physically deletes table from the database.
DROP TRIGGER	Deletes a trigger.
DROP VIEW	Deletes a view.

GRANT (database authority)	Grants database authority or privileges
GRANT (table privileges)	Grants one or more specified privileges for a table or view.
GRANT EXECUTE ON	Grants execute privilege on stored procedures and external functions to other users.
INSERT .	Inserts one or more rows into an existing table
INSTALL DATABASE	Puts a database on the network, making it accessible to users.
LABEL	Adds or changes labels in catalog descriptions
LOAD	Loads one or more tables into a database.
LOCK DATABASE	Places an exclusive lock on the database, preventing connections from other users.
PROCEDURE :	Creates a procedure
REVOKE	Revokes database authority or privileges.
REVOKE EXECUTE ON	Revokes users execute privilege on a stored procedure or external function.
ROLLBACK	Terminates a logical unit of work and backs out database changes made during the last transaction.
ROWCOUNT	Obtains the number of rows in a table.
SAVEPOINT	Assigns a checkpoint within a transaction.
SELECT	Queries tables or views.
SET DEFAULT STOGROUP	Specifies the default storage group.
START AUDIT	Starts a database audit.
STOP AUDIT	Stops a database audit.
UNLOAD	Unloads a database to an external file.
UNLOCK DATABASE	Releases the exclusive lock on the database from the LOCK DATABASE command
UPDATE	Updates the values of columns in a table or view.
UPDATE STATISTICS	Updates the statistics for a database, table, or index

❑ **Check Your Progress – 2 :**

1. Changes an external function definition.

   a. ALTER EXTERNAL FUNCTION

   b. ALTER TRIGGER

2. Enables and disables triggers defined on tables.

   a. ALTER TRIGGER   b. ALTER EXTERNAL FUNCTION

3. Creates a storage group.

   a. CREATE STOGROUP   b. GENERATE STOGROU

4. Physically deletes a database area.

   a. DROP DBAREA   b. DOWN DBAREA

5. Starts a database audit.

   a. START AUDIT   b. AUDIT START

3.3 Let Us Sum Up :

In the above discussion even the various SQL* DBA commands have been discussed in very detail.

3.4 Answers for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a	**2 :** a	**3 :** a	**4 :** a	**5 :** a

❑ **Check Your Progress 2 :**

1 : a	**2 :** a	**3 :** a	**4 :** a	**5 :** a

3.5 Glossary :

DBA : Database Administration

3.6 Assignment :

Throw some light on the functions of a database administrator

3.7 Activities :

Discuss in detail the role of a database administrator in an organisation

3.8 Case Study :

Discuss the role of database administrator when the database is corrupted.

3.9 Furthers Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw–Hill Publication.
3. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
4. Database Management System by Raghu Ramkrishnan – Tata McGraw–Hill Publication.
5. SQL, PL/SQL: The Programming Language Oracle – Ivan Bayross – BPB Publication.

BLOCK SUMMARY :

In this block in very detail about SQL* Plus, we learnt about on how can we use the SQL* Plus program in conjunction with the SQL database language and its procedural language extension, PL/SQL, This block has taught us that the SQL database language allows to store and retrieve data in Oracle. PL/SQL allows you to link several SQL commands through procedural logic. SQL* Plus enables you to execute SQL commands and PL/SQL blocks, and to perform many additional tasks as well. This ?lock has taught us that the SQL database language allows storing and retrieving data in Oracle. PL/SQL allows you to link several SQL commands through procedural logic. SQL* Plus enables you to execute SQL commands and PL/SQL blocks, and to perform many additional tasks as well.

The main aim of this block was to give a basic introduction to the students of computers science about SQL.

BLOCK ASSIGNMENT :

❖ **Short Questions :**

Write short notes on :

1. DBA
2. SQL DBA Commands reference for

   (1) Create (2) Alter (3) Drop

❖ **Long Questions :**

1. Give a brief note on SQL DBA.

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	1	2	3
No. of Hrs.			

2. Please give your reactions to the following items based on your reading of the block :

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	
Language and Style	☐	☐	☐	☐	
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	
Conceptual Clarity	☐	☐	☐	☐	
Check your progress Quest	☐	☐	☐	☐	
Feed back to CYP Question	☐	☐	☐	☐	

3. Any other Comments

....

....

....

....

....

....

....

....

Dr. Babasaheb Ambedkar Open University Ahmedabad

BCAR 302

RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)

BLOCK 2 : INTERACTIVE SQL AND ORACLE

INTERACTIVE SQL AND ORACLE

Block Introduction :

This unit provides an introduction to Interactive SQL & Oracle. It provides an overview of how use Oracle and demonstrates this with the help of various examples.

This block will give the students a basic idea of what SQL and Oracle are. Various functions and commands have been discussed here in this block in a very brief and interesting way. The main topics that have been covered in this block are queries writing, tables and view options, decode statement and insert, update and delete commands. Apart from this the block will even throw some light on the limited portion of interactive SQL and the uses of various SQL functions The writer has tried his best to concise the topics using the most easy language to help the student to understand the topic.

The aim of this block is to enable the reader understand the basic concepts of query writing and sub queries. The writer has tried his best to detail the create statement, apart from this the writing queries has even been discussed by him in detail. On the other hand he has even disucussed the table and view option. The decode statement, insert, update and delete commands have been discussed in detail with sufficient examples. The interactive SQL portion has even been discussed here in this block in a very detail in unit 2. The author has explained the various data types briefly. Apart from this the objective of this block includes detailing the student's about the various Oracle functions which has been discussed in unit 3. Here under oracle functions the topics covered are the library functions as well as the uses of various SQL functions in a very detail with the help of sufficient and suitable examples.

Block Objectives :

After learning this block, you will be able to understand :

- The process of Query writing.
- The functions of various commands
- Interactive SQL and various types of data in SQL
- Oracle and its functions
- Library functions of oracle

Block Structure :

Unit 04

QUERY WRITING AND SUB QUERIES

UNIT STRUCTURE

4.0 Learning Objectives :

After learning this unit, you will be able to understand :

- Query writing.
- The decode statement
- Insert, Update and delete commands

4.1 Introduction :

A query is a request for data results, for action on data, or for both. You can use a query to answer a simple question, to perform calculations, to combine data from different tables, or even to add, change, or delete table data. Queries that you use to retrieve data from a table or to make calculations are called select queries. Queries that add, change, or delete data are called action queries.

Subqueries provide a powerful means to combine data from two tables in to a single result. Subqueries are sometimes called nested queries. As the name implies, subqueries contain one or more queries, one inside the other.

Subqueries are very versatile and that can make them some what hard to understand. For most cases a subquery can be used anywhere you can use an expression or table specification.

For example, you can use subqueries in the SELECT, FROM, WHERE, or HAVING clauses. Depending on its use, a subquery may return one or more rows, or in other cases, such as when used in a SELECT clause, be restricted to returning a single value.

4.2 The Create Statements :

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with an example below.

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. You can check complete details at Create Table Using another Table.

The CREATE TABLE statement is used to create a table in a database.

Tables are organized into rows and columns; and each table must have a name.

➢ **SQL CREATE TABLE Syntax :**

```
CREATE TABLE table_name
(
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
....
);
```

The column_name parameters specify the names of the columns of the table.

The data_type parameter specifies what type of data the column can hold (e.g. varchar, integer, decimal, date, etc.).

The size parameter specifies the maximum length of the column of the table. **Illustration**

Now we want to create a table called ?Persons? that contains five columns:

PersonID, LastName, FirstName, Address, and City.

We use the following CREATE TABLE statement:

Example

```
CREATE TABLE Persons
(
PersonIDint,
LastNamevarchar(255),
FirstNamevarchar(255),
Address varchar(255),
City varchar(255)
);
```

❑ **Check Your Progress – 1 :**

1. ________ TABLE is the keyword telling the database system what you want to do.

   a. CREATE   b. SHOW

2. In ________ comes the list defining each column in the table and what sort of data types it is.

   a. brackets   b. semi colons

3. A copy of an existing table can be created using a combination of the CREATE TABLE statement and the ________ statement.

   a. SELECT   b. SHOW

4. The ________ TABLE statement is used to create a table in a database.

   a. CREATE   b. SHOW

5. The ________ parameter specifies the maximum length of the column of the table.

   a. Size   b. scope

4.3 Writing Queries :

Before you get started, it's important to become accustomed to your database and its hierarchy. If you have multiple databases of data, you'll need to zero in on the location of the data you want to work with.

For example, let's pretend we're working with multiple databases about people in the United States. Type in the query ―SHOW DATABASES;‖. Our results may show that you have a couple of databases for different locations, including one for New England.

Within your database, you'll have different tables containing the data you want to work with. Using the same example above, let's say we want to find out which information is contained in one of the databases. If we use the query ―SHOW TABLES in NewEngland;‖, we'll find that we have tables for each state in New England: people_connecticut, people_maine, people_massachusetts, people_newhampshire, people_rhodeisland, and people_vermont.

Finally, you need to find out which fields are in the tables. Fields are the specific pieces of data that you can pull from your database. For example, if you want to pull someone's address, the field name may not just be ―address‖ --it may be separated into address_city, address_state, address_zip. In order to figure this out, use the query ―Describe people_massachusetts;‖. That will provide a list of all of the data that you can pull using SQL.

Let's do a quick review of the hierarchy using our New England example :

Our database is : NewEngland.

Our tables within that database are : people_connecticut, people_maine, people_massachusetts, people_newhampshire, people_rhodeisland, and people_vermont.

Our fields within the people_massachusetts table include : address_city, address_state, address_zip, hair_color, first_name, and last_name.

Now, to learn how to write a simple SQL query, let's use the following example :

Who are the people who have red hair in Massachusetts and were born in 2003 organized in alphabetical order ?

- **SELECT :**

SELECT chooses the fields that you want displayed in your chart. This is the specific piece of information that you want to pull from your database. In the example above, we want to find the people who fit the rest of the criteria.

Here is our SQL query :

SELECT

first_name,

last_name

- **FROM :**

FROM pinpoints the table that you want to pull the data from. In the earlier section, we found that there were six tables for each of the six states in NewEngland : people_connecticut, people_maine, people_massachusetts, people_newhampshire, people_rhodeisland, and people_vermont. Because we're looking for people in Massachusetts specifically, we'll pull data from that specific table

Here is our SQL query :

SELECT

first_name,

last_name

FROM

people_massachusetts

- **WHERE :**

WHERE allows you to filter your query to be more specific. In our example, we want to filter our query to include only people with red hair who were born in 2003. Let's start with the red hair filter.

Here is our SQL query :

SELECT

first_name,

last_name

FROM

people_massachusetts

WHERE

hair_color = ―red‖

hair_color could have been part of your initial SELECT statement if you'd wanted to look at all of the people in Massachusetts along with their specific hair color. But if you want to filter to see only people with red hair, you can do so in the WHERE statement.

- **AND :**

AND allows you to add additional criteria to your WHERE statement. Remember, we want to filter by people who had red hair in addition to people who were born in 2003. Since our WHERE statement is taken up by the red hair criteria, how can we filter by a specific year of birth as well ?

That's where the AND statement comes in. In this case, the AND statement is a date property -- but it doesn't necessary have to be. (**Note :** Be to check the format of your dates with your product team to make sure it is in the correct format.)

Here is our SQL query :

SELECT

first_name,

last_name

FROM

people_massachusetts

WHERE

hair_color = ―red‖

AND

birth_date BETWEEN ‗2003–01–01' AND ‗2003–12–31'

➢ **ORDER BY :**

When you create SQL queries, you shouldn't have to export the data to Excel. The calculation and organization should be done within the query. That's where the ―ORDER BY‖ and ―GROUP BY‖ functions come in. First, we'll look at our SQL queries with the ORDER BY and then GROUP BY functions, respectively. Then, we'll take abrief look at the difference between the two.

Your ORDER BY clause will allow you to sort by any of the fields that you have specified in the SELECT statement. In this case, let's order by last name.

Here is our SQL query :

SELECT

first_name,

last_name

FROM

people_massachusetts

WHERE

hair_color = ―red‖

AND

birth_date BETWEEN ‗2003–01–01' AND ‗2003–12–31'

ORDER BY

last_name

;

➢ **GROUP BY :**

―GROUP BY‖ is similar to ―ORDER BY,‖ but it will aggregate data that has similarities. For example, if you have any duplicates in your data, iyou can use ―GROUP BY‖ to count the number of duplicates in your fields.

Here is your SQL query :

SELECT

first_name,

last_name

FROM

people_massachusetts

WHERE

hair_color = ―red‖

AND

birth_date BETWEEN ‗2003–01–01' AND ‗2003–12–31'

GROUP BY

last_name

;

➢ **ORDER BY VS. GROUP BY :**

To clearly show you the difference between an ―ORDER BY‖ statement and a ―GROUP BY‖ statement, let's step outside our Massachusetts example briefly to look at a very simple dataset. Below is a list of four employees' ID numbers and names.

order by group by

If we were to use an ORDER BY statement on this list, the names of the employees would get sorted in alphabetical order. The results would look like this:

order by

If we were to use a GROUP BY statement, the employees would be counted based on the number of times they appeared in the initial table.

Group by

With me so far ? Okay. Let's return to the SQL query we've been creating about red–haired people in Massachusetts who were born in 2003.

➢ **LIMIT :**

Depending on the amount of data you have in your database, it may take a long time to run the queries. It can be frustrating if you find yourself waiting a long time to run a query that you didn't really want to begin with. If you want to test our query, the LIMIT function is a great one to use because it allows you to limit the number of results you get.

For example, if we suspect there are millions of people who have red hair in Massachusetts, we may want to test out our query using LIMIT before we run it in full to make sure we're getting the information we want. Let's say, for instance, we only want to see the first 100 people.

Here is our SQL query :

SELECT

first_name,

last_name

FROM

people_massachusetts

WHERE

hair_color = —red||

AND

birth_date BETWEEN ‗2003–01–01' AND ‗2003–12–31'

ORDER BY

last_name

LIMIT

100

;

That's it for the basics !

Feeling good ? Here are a few other ways to take your SQL queries up a notch.

Bonus : Advanced SQL Tips

Now that you have mastered how to create a SQL query, let's walk through some other tricks that you can use to take it up a notch, starting with the asterisk.

*

When you add an asterisk to one of your SQL queries, it tells the query that you want to include all the columns of data in your results. In the example we've been using, we've only had two column names: first_name and last_name. But let's say we had 15 columns' worth of data that we want to see in our results -- it would be kind of a pain to type out all 15 column names in the SELECT statement.

Instead, if you replace the names of those columns with an asterisk, the query will know to pull all of the columns in to the results.

Here's what the SQL query would look like :

SELECT *FROM

people_massachusetts

WHERE

hair_color = —red||

AND

birth_date BETWEEN ‗2003–01–01' AND ‗2003–12–31'

ORDER BY

last_name

LIMIT

100

;

- **LAST 30 DAYS :**

Once I started using SQL regularly, I found that one of my go–to queries involved trying to find which people took an action or fulfilled a certain set of criteria within the last 30 days. Since this type of query was so useful for me, I wanted to share that capa?ility with you.

Let's pretend today is December 1, 2014. You could create these parameters by making the birth_date span between November 1, 2014 and November 30, 2014. That SQL query would look like this:

```
SELECT
first_name,
last_name
FROM
people_massachusetts
WHERE
hair_color = —red||
AND
birth_date BETWEEN ‗2014–11–01' AND ‗2014–11–30'
ORDER BY
last_name
LIMIT
        100
;
```

But that would require thinking about which dates cover the last 30 days, and it would mean you'd have to constantly update this query. Instead, to make the dates automatically span the last 30 days no matter which day it is, you can type this under AND : birth_date>= (DATE_SUB(CURDATE(),INTERVAL 30.

(**Note :** You'll want to double–check this syntax with your product team because it may differ based on the software you use to pull your SQL queries.)

Your SQL query would therefore look like this :

```
SELECT
first_name,
last_name
FROM
people_massachusetts
WHERE
hair_color = —red||
AND
birth_date>= (DATE_SUB(CURDATE(),INTERVAL 30))
ORDER BY
last_name
LIMIT
        100
;
```

➢ **COUNT :**

In some cases, you may want to count the number of times that a criterium of a field appears. For example, let's say you want to count the number of times the different hair colors appear for the people you are tallying up from Massachusetts. In this case, COUNT will come in handy so you don't have to manually add up the number of people who have different hair colors or export that information to Excel.

Here's what that SQL query would look like :

```
SELECT
hair_color,
COUNT(hair_color)
FROM
people_massachusetts
AND
birth_date BETWEEN ‗2003–01–01' AND ‗2003–12–31'
GROUP BY
hair_color
;
```

➢ **JOIN :**

There may be a time where you need to access information from two different tables in one SQL query. In SQL, you can use a JOIN clause to do this. (For those of you familiar with Excel formulas, this is similar to how you would use the VLOOKUP formula when you need to combine information from two different sheets in Excel.)

For example, let's say we have one table that has data of all Massachusetts residents' user IDs and their birthdates. Let's say we also have an entirely separate table that has data of all Masachusetts residents' user IDs and their hair color. If we want to figure out the hair color of Massachusetts residents born in the year 2003, we'd need to access information from both tables and combine them. This works because both tables share a matching column: the Massachusetts residents' user IDs.

Because we're calling out fields from two different tables, our SELECT statement is also going to change slightly. Instead of just listing out the fields we want to include in our results, we'll need to specify which table they're coming from. (Note: The asterisk function may come in handy here so your query includes both tables in your results.)

To specify a field from a specific table, all we'd have to do is combine the name of the table with the name of the field. For example, our SELECT statement would say ―table.field‖ -- with the period separating the table name and the field name.

Let's take a look at what this looks like in action.

We're assuming a few things in this case :

The Massachusetts birthdate table includes the following fields : first_name, last_name, user_id, birthdate

The Massachusetts hair color table includes the following fields: user_id, hair_color

Your SQL query would therefore look like :

```
SELECT
birthdate_massachusetts.first_name,
birthdate_massachusetts.last_name
FROM
birthdate_massachusetts JOIN haircolor_massachusetts USING (user_id)
WHERE
hair_color = —red||
AND
birth_date BETWEEN ‗2003–01–01' AND ‗2003–12–31'
ORDER BY
last_name
;
```

This query would join the two tables using the field —user_id|| which appears in both the birthdate_massachusetts table and the haircolor_massachusetts table. You would then be able to see a table of people born in 2003 that have red hair.

❑ **Check Your Progress – 2 :**

1. ________ chooses the fields that you want displayed in your chart.

   a. SELECT b. FROM

2. ________ pinpoints the table that you want to pull the data from.

   a. FROM b. SELECT

3. ________ allows you to filter your query to be more specific.

   a. WHERE b. FROM

4. ________ allows you to add additional criteria to your WHERE statement.

   a. AND b. FROM

5. Depending on the amount of ________ you have in your database, it may take a long time to run the queries.

   a. Data b. Table

4.4 Tables and Views :

➢ **Views :**

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following :

Structure data in a way that users or classes of users find natural or intuitive.

Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.

Summarize data from various tables which can be used to generate reports.

➢ **Creating Views :**

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables, or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows :

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

You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL SELECT query.

Example :

Consider the CUSTOMERS table having the following records:

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

Now, following is the example to create a view from CUSTOMERS table. This view would be used to have customer name and age from CUSTOMERS table:

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

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

```
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

The WITH CHECK OPTION :

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following is an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION :

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM CUSTOMERS
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View :

A view can be updated under certain conditions :

The SELECT clause may not contain the keyword DISTINCT.

The SELECT clause may not contain summary functions.

The SELECT clause may not contain set functions.

The SELECT clause may not contain set operators.

The SELECT clause may not contain an ORDER BY clause.

The FROM clause may not contain multiple tables.

The WHERE clause may not contain subqueries.

The query may not contain GROUP BY or HAVING.

Calculated columns may not be updated.

All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So if a view satisfies all the above–mentioned rules then you can update a view. Following is an example to update the age of Ramesh :

```
SQL > UPDATE CUSTOMERS_VIEW
SET AGE = 35
WHERE name=‗Ramesh';
```

This would ultimately update the base table CUSTOMERS and same would reflect in the view itself. Now, try to query base table, and SELECT statement would produce the following result :

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	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

Inserting Rows into a View :

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here we can not insert rows in CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view; otherwise you can insert rows in a view in similar way as you insert them in a table.

Deleting Rows into a View :

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE= 22.

```
SQL > DELETE FROM CUSTOMERS_VIEW
WHERE age = 22;
```

This would ultimately delete a row from the base table CUSTOMERS and same would reflect in the view itself. Now, try to query base table, and SELECT statement would produce the following result :

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	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	Muffy	24	Indore	10000.00

Dropping Views :

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple as given below :

DROP VIEW view_name;

Following is an example to drop CUSTOMERS_VIEW from CUSTOMERS table :

DROP VIEW CUSTOMERS_VIEW;

❑ **Check Your Progress – 3 :**

1. A ________ is nothing more than a SQL statement that is stored in the database with an associated name.

   a. view b. file

2. Database views are created using the ________ statement.

   a. CREATE VIEW b. DROP VIEW

3. ________ of data can be inserted into a view

   a. Rows b. Column

4.5 Let Us Sum Up :

In the above unit we discussed the various query writing techniques. A detailed discussion was made and the various techniques studied are

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.A view is nothing more than a SQL statement that is stored in the database with an associated name. It can contain all rows of a table or select rows from a table. The DECODE function is a typical remnant from the days that Oracle SQL did not yet support CASE expressions. The INSERT command is used to add rows to a table. The column values of existing rows in your tables with the UPDATE command.

The simplest data manipulation command is DELETE, It operates at the table level, and you use the WHERE clause to restrict the set of rows you want to delete from the table.The DROP TABLE command not only removes the contents of the table, but also the table itself.

4.6 Answers for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a **2 :** a **3 :** a

❑ **Check Your Progress 2 :**

1 : a **2 :** a

❑ **Check Your Progress 3 :**

1 : a **2 :** a **3 :** a

4.7 Glossary :

1. **SQL :** Structured Query Language.

4.8 Assignment :

Explain the Create Statement using suitable example.

4.9 Activities :

What do you understand by query and subqueries ? Explain its utility ?

4.10 Case Study :

Discuss the creation of views by giving suitable example.

4.11 Further Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw – Hill Publication.
3. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
4. Database Management System by Raghu Ramkrishnan – Tata McGraw – Hill Publication.
5. SQL, PL/SQL : The Programming Language Oracle – Ivan Bayross – BPB Publication.

Unit 05 QUERY WRITING

UNIT STRUCTURE

5.0 Learning Objectives :

After learning this unit, you will be able to understand :

- Query writing.
- The decode statement
- Insert, Update and delete commands

5.1 Introduction :

A lengthy conversation was held, and the many strategies examined were discussed in detail. The keyword CREATE TABLE tells the database system what you want to accomplish. You want to make a new table in this scenario. The CREATE TABLE statement is followed by the table's unique name or identifier. A view is nothing more than a SQL statement with a name that is stored in the database. It can contain all of a table's rows or only a few of them. The DECODE function is a holdover from the days before Oracle SQL supported CASE expressions. To add rows to a table, use the INSERT command. With the UPDATE command, you can change the column values of existing rows in your tables.

5.2 The Decode Statement :

The DECODE function is a typical remnant from the days that Oracle SQL did not yet support CASE expressions. There are three good reasons not to use DECODE anymore :

- DECODE function expressions are quite difficult to read.
- DECODE is not part of the ANSI/ISO SQL standard.
- CASE expressions are much more powerful.

For completeness, and because you may encounter the DECODE function in legacy OracleSQL programs, illustration shows a query where the DECODE function is used in the SELECTclause and in the ORDER BY clause.

Illustration : Using the DECODE Function

```
SQL> select job, ename
decode(greatest(msal,2500)
2500,'cheap','expensive') as class
from employees
where bdate< date ‗1964–01–01'
order by decode(job,'DIRECTOR',1,'MANAGER',2,3);
JOB ENAME CLASS
---------- ---------- ----------
DIRECTOR KING expensive
MANAGER BLAKE expensive
SALESREP ALLEN cheap
SALESREP WARD cheap
ADMIN MILLER cheap
TRAINER FORD expensive
TRAINER SCOTT expensive
SALESREP MARTIN cheap
SQL>
```

❑ **Check Your Progress – 1 :**

1. ________ function expressions are quite difficult to read.

   a. DECODE   b. ENCODE

2. DECODE is ________ part of the ANSI/ISO SQL standard.

   a. Not a   b. a

3. CASE expressions are much ________ powerful.

   a. More   b. less

5.3 Insert :

You can use the INSERT command to add rows to a table. Along with the standard INSERTcommand, Oracle SQL also supports a multitable version.

➢ **Standard INSERT Commands :**

The standard INSERT command supports the following two ways to insert rows :

- Use the VALUES clause, followed by a list of column values (betweenparentheses).

  This method allows you to insert only one row at a time per execution of the INSERTcommand.

- Formulate a subquery, thus using existing data to generate new rows.

  Both alternatives are shown in the syntax diagram in Figure 1.1

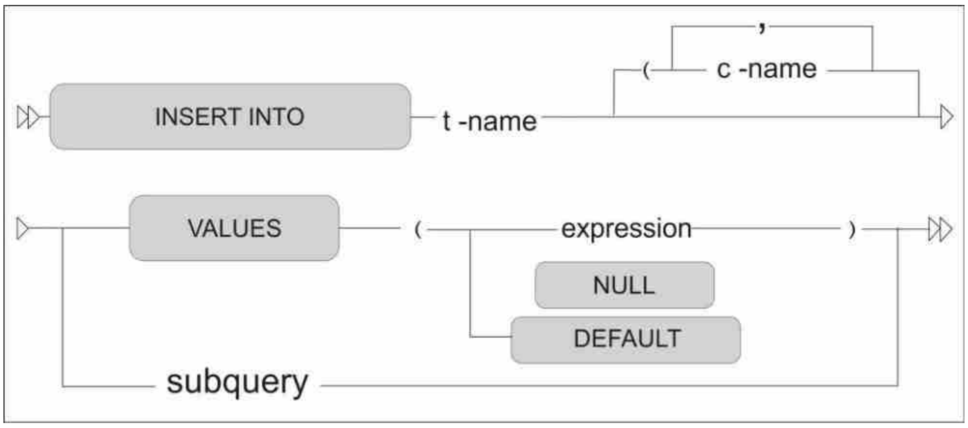


Fig. 5.1 INSERT Command Syntax Diagram

If you know all of the table columns, including the internal physical order in which theyare presented by the SQL* Plus DESCRIBE command, you don't need to specify column namesafter the table name in the INSERT command. If you omit column names, you must provideprecisely enough values and specify them in the correct order.

In the VALUES clause, you can specify a comma–separated list of literals or an expression.

You can use the reserved word NULL to specify a null value for a specific column. You can alsospecify the reserved word DEFAULT to instruct the Oracle DBMS to insert the default value associatedwith the corresponding column. These default values are part of the table definition, stored in the data dictionary. If you don't specify a value for a specific column in your INSERTstatement, there are two possibilities:

- If the column has an associated DEFAULT value, the Oracle DBMS will insert that value.
- If you did not define a DEFAULT value for the column, the Oracle DBMS inserts a nullvalue (provided, of course, that the column allows null values).

The second way of using the INSERT command fills a table with a subquery. There areno special constraints for these subqueries, as long as you make sure they produce the rightnumber of values of the right datatype. You can even use a subquery against the table intowhich you are inserting rows. This sound like a strange approach; however, INSERT INTO XSELECT * FROM X is one of the fastest methods to fill a table, provided you don't have unique orprimary key constraints.

Listing 5.1 shows four INSERT statement examples: three using the VALUES clause and oneusing the subquery method.

Listing 5.1 Four INSERT Command Examples

```
SQL> insert into departments -- Example 1

2 values (90,'SUPPORT','SEATTLE', NULL);

1 row created.

SQL> insert into employees (empno, ename, init, ?date, msal, deptno)
-- Example 2

2 values (7001,'ZOMBIE','ZZ', trunk (sysdate), 0, DEFAULT);
```

```
1 row created.
SQL> select * from employees where empno = 7001;
EMPNO ENAME INIT JOB MGR BDATE MSAL COMM DEPTNO
----- ----- ----- ----- ----- ----- ----- ----- ----- -----
7001 ZOMBIE ZZ 15–SEP–2004 0 10
SQL> insert into departments (dname, location, deptno) — Example 3
2 values (‗CATERING','ORLANDO', 10);
insert into departments(dname,location,deptno)
*
ERROR at line 1:
ORA–00001: unique constraint (BOOK.D_PK)
violated SQL> insert into salgrades -- Example 4
2 select grade + 5
3 , lowerlimit + 2300
4 , least(9999, upperlimit + 2300)
5,500
6 from salgrades;
5 rows created.
SQL> rollback;
Rollback complete.
SQL>
```

The examples work as follows :

- The first example inserts a new department 90 without specifying column names.
- It also shows how you can insert a null value with the reserved word NULL.
- The second example shows how you can use DEFAULT to assign the default department number to a new employee. The default value for the DEPTNO column of the EMPLOYEES table is 10, as you can see inListing 6–1.
- The third example shows a violation of a primary key constraint; department 10 already exists.
- The fourth example shows how you can use a subquery to insert rows with the INSERT command. It uses the LEAST to avoid constraint violations. The first argument (9999) ensures that the upper limits will never become greater than 9999.

At the end of Listing 1.1, we use ROLLBACK to undo our changes.

❑ **Check Your Progress – 2 :**

1. You can use the ________ command to add rows to a table.

   a. INSERT b. DELETE

2. The second way of using the INSERT command fills a table with a ________.

   a. Subquery b. query

5.4 Update :

You can change column values of existing rows in your tables with the UPDATE command. As shown in the syntax diagram in Figure 1.2, the UPDATE command has three main components:

- **UPDATE :** The table you want to update
- **SET :** The change you want to apply
- **WHERE :** The rows to which you want to apply the change

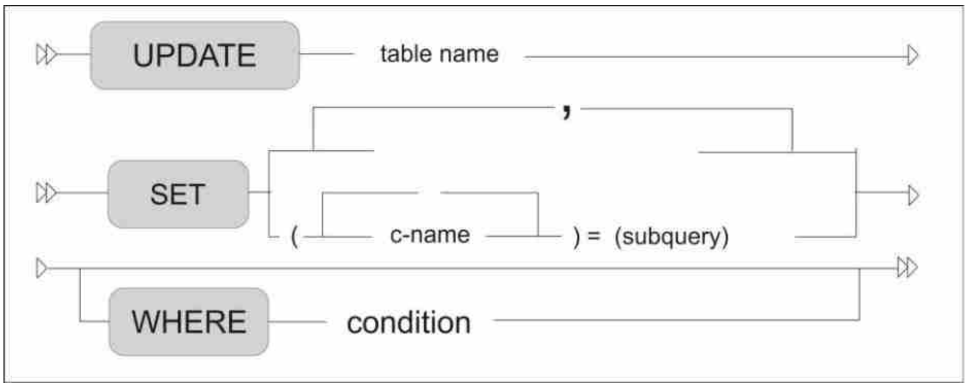


Fig. 5.2 UPDATE Command Syntax Diagram

If you omit the optional WHERE clause, the change is applied to all rows of the table.This illustrates the fact that the UPDATE command operates at the table level, so you need the WHERE clause as the relational restriction operator to limit the scope of the UPDATE command toa subset of the table.

As you can see from Figure 5.2 the SET clause offers two alternatives :

- You can specify a comma–separated list of single–column changes. With this approach, you can use the DEFAULT keyword as an expression. This allows you to change columndefault values in the data dictionary at any point in time without the need to changethe UPDATE commands in your applications.
- You can drive the change with a subquery. The subquery must provide the right numberof values for the list of column names specified between the parentheses. Of course, the data types should also match, or the Oracle DBMS should at least be able to convertvalues to the appropriate datatypes on the fly.

The first approach is illustrated in Listing 5.2, and the second approach is shown in Listing 5.2

Listing 5.2 UPDATE Command Example

```
SQL> update employees
2 set job = ?SALESREP'
3 , msal = msal – 500
4 , comm = 0
5 , deptno = 30
```

6 where empno = 7876;

1 row updated. SQL> rollback; Rollback complete. SQL>

Listing 5.2 UPDATE Command Example Using a Subquery SQL> update registrations

2 set evaluation = 1

3 where (course, begindate)

4 in (select course, begindate

5 from offerings

6 where location = ‗CHICAGO');

3 rows updated.

SQL> rollback;

Rollback complete.

SQL>

As with the INSERT examples in Listing 5.6–1, in both of these listings, we use the ROLLBACKcommand to undo any changes made.

❑ **Check Your Progress – 3 :**

1. You can change column values of existing rows in your tables with the ________ command.

   a. UPDATE   b. EDIT

2. You can specify a ________ –separated list of single–column changes.

   a. comma   b. Colon

3. The ________ must provide the right number of values for the list of column names specified between the parentheses.

   a. Subquery   b. Query

5.5 Delete :

The simplest data manipulation command is DELETE, as shown in the syntax diagram in Figure 5.3 This command also operates at the table level, and you use the WHERE clause torestrict the set of rows you want to delete from the table. If you omit the WHERE clause, the DELETE command results in an empty table.

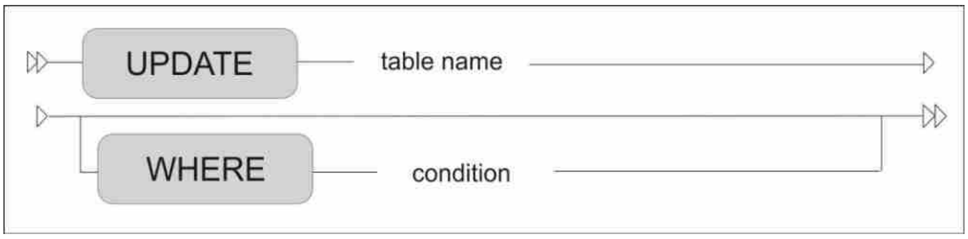


Fig. 5.3 DELETE Command Syntax Diagram

Note the difference between the following two commands:

SQL> drop table departments;

SQL> delete from departments;

The DROP TABLE command not only removes the contents of the table, but also the tableitself, including all dependent objects/structures such as indexes and privileges. DROP TABLE is adata definition (DDL) command. The DELETE command does not change the database structure, but only the contents–it is a data manipulation (DML) command. Moreover, the effects of a DROP TABLE command cannot be undone with a ROLLBACK command, as opposed to the effects ofa DELETE command, which can. Listing 5.8–1 shows how you can delete a salary grade.

Listing 5.3 Example of a DELETE Command

```
SQL> delete from salgrades
2 where grade = 5;
1 row deleted.
SQL> rollback;
Rollback complete.
SQL>
```

To illustrate the fact that you can also use subqueries in the FROM clause of the DELETE statement, Listing 5.3 shows an alternative formulation for the same DELETE statement. Again, we use the ROLLBACK command to undo our changes.

Listing 5.3 Alternative DELETE Command, Using a Subquery

```
SQL> delete from (select *
2 from salgrades
3 where grade = 5);
1 row deleted.
SQL> rollback;
Rollback complete.
SQL>
```

In this case, there are no obvious advantages to using a subquery over using a regular DELETE statement. However, the subquery syntax also works (under certain conditions) with more complicated subqueries, opening up some very interesting possibilities. Deleting rows may seem rather straight forward, but you might encounter complications due to constraint violations. The same is true for the UPDATE and INSERT commands, by theway.

❑ **Check Your Progress – 4 :**

1. The simplest data manipulation command is ________.

   a. DELETE b. ROLL BACK

2. The ________ command not only removes the contents of the table, but also the table itself.

   a. DROP TABLE b. ROLL BACK

3. DROP TABLE command cannot be undone with a ________ command.

   a. ROLLBACK b. DELETE

5.6 Let Us Sum Up :

In the above unit we discussed the various query writing techniques. A detailed discussion was made and the various techniques studied are CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.A view is nothing more than a SQL statement that is stored in the database with an associated name. It can contain all rows of a table or select rows from a table. The DECODE function is a typical remnant from the days that Oracle SQL did not yet support CASE expressions. The INSERT command is used to add rows to a table. The column values of existing rows in your tables with the UPDATE command.

The simplest data manipulation command is DELETE, It operates at the table level, and you use the WHERE clause to restrict the set of rows you want to delete from the table.The DROP TABLE command not only removes the contents of the table, but also the table itself.

5.7 Answers for Check Your Progress :

- **Check Your Progress 1 :**

  1 : a **2 :** a **3 :** a

- **Check Your Progress 2 :**

  1 : a **2 :** a

- **Check Your Progress 3 :**

  1 : a **2 :** a **3 :** a

- **Check Your Progress 4 :**

  1 : a **2 :** a **3 :** a

5.8 Glossary :

1. SQL : Structured Query Language.

5.9 Assignment :

Explain the DELETE Statement using suitable example.

5.10 Activities :

What do you understand by INSERT and UPDATE Query ? Explain its utility ?

5.11 Case Study :

Discuss the decode statement by giving suitable example.

5.12 Further Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw – Hill Publication.
3. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
4. Database Management System by Raghu Ramkrishnan – Tata McGraw – Hill Publication.
5. SQL, PL/SQL: The Programming Language Oracle – Ivan Bayross – BPB Publication.

Unit 06

INTERACTIVE SQL

UNIT STRUCTURE

6.0 Learning Objectives :

After learning this unit, you will be able to understand :

- Interactive SQL and its command file
- The various data types in SQL

6.1 Introduction :

Interactive SQL (dbisql) is a utility for entering SQL statements. If you use Interactive SQL to work with your database schema, instead of executing the SQL statements one at a time, build up the set of commands in a dbisql command file. Then you can execute this file in dbisql to build the database.

The definitions of the database objects form the database schema. You can think of the schemaas an empty database. The SQL statements for creating and modifying schemas are called the data definition language (DDL).

Note : Only one user at a time can perform DDL statements on a table. IQ locks a table during DDL operations on it. Users may, however, perform DDL on other objects in the same database at the same time.

If you use a tool other than Interactive SQL, all the information in these topics concerning SQL statements still applies.

Interactive SQL Command File

An Interactive SQL command file is a text file with semicolons placed at the end of commands as shown below.

```
CREATE TABLE t1 (..);
CREATE TABLE t2 (..);
CREATE LF INDEX i2 ON t2 (..);
..
```

An Interactive SQL command file usually carries the extension .sql. To execute a command file, either paste the contents of the file into the Interactive

SQL command window (if the file has less than 500 lines) or enter a command that reads the file into the command window. For example, the READ statement:

readmakedb

Reads the Interactive SQL commands in the file makedb.sql.

6.2 Data Types :

SQL data type is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL. You would use these data types while creating your tables. You would choose a particular data type for a table column based on your requirement. SQL Server offers six categories of data types for your use :

Exact Numeric Data Types :

DATA TYPE	FROM	TO
Bigint	–9,223,372,036,854,775,808	9,223,372,036,854,775,807
Int	–2,147,483,648	2,147,483,647
Smallint	–32,768	32,767
Tinyint	0	255
Bit	0	1
Decimal	–10^38 +1	10^38 –1
Numeric	–10^38 +1	10^38 –1
Money	–922,337,203,685,477.5808	+922,337,203,685,477.5807
Smallmoney	–214,748.3648	+214,748.3647

Approximate Numeric Data Types :

DATA TYPE	FROM	TO
Float	–1.79E + 308	1.79E + 308
Real	–3.40E + 38	3.40E + 38

Date and Time Data Types :

DATA TYPE	FROM	TO
Datetime	Jan 1, 1753	Dec 31, 9999
Smalldatetime	Jan 1, 1900	Jun 6, 2079
Date	Stores a date like June 30, 1991	
Time	Stores a time of day like 12:30 P.M.	

Note : Here, date time has 3.33 milli seconds accuracy where as small date time has 1 minute accuracy.

Character Strings Data Types :

DATA TYPE	FROM	TO
Char	Char	Maximum length of 8,000 characters. (Fixed length non–Unicode characters)
Varchar	Varchar	Maximum of 8,000 characters. (Variable–length non–Unicode data).
varchar(max)	varchar(max)	Maximum length of 231 characters, Variable–length non–Unicode data (SQL Server 2005 only).
Text	text	Variable–length non–Unicode data with a maximum length of 2,147,483,647 characters.

Unicode Character Strings Data Types :

DATA TYPE	Description
Nchar	Maximum length of 4,000 characters. (Fixed length Unicode)
Nvarchar	Maximum length of 4,000 characters. (Variable length Unicode)
nvarchar(max)	Maximum length of 231characters (SQL Server 2005 only). (Variable length Unicode)
Ntext	Maximum length of 1,073,741,823 characters. (Variable length Unicode)

Binary Data Types :

DATA TYPE	Description
Binary	Maximum length of 8,000 bytes (Fixed–length binary data)
Varbinary	Maximum length of 8,000 bytes. (Variable length binary data)
varbinary(max)	Maximum length of 231 bytes (SQL Server 2005 only). (Variable length Binary data)
Image	Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)

Misc Data Types :

DATA TYPE	Description
sql_variant	Stores values of various SQL Server–supported data types, except text, ntext, and timestamp.
Timestamp	Stores a database–wide unique number that gets updated every time a row gets updated
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML data. You can store xml instances in a column or a variable (SQL Server 2005 only).
cursor	Reference to a cursor object
table	Stores a result set for later processing

- **Check Your Progress – 1 :**

1. Each column, variable and expression has ________ data type in SQL.

   a. Related b. Unrelated

6.3 Let Us Sum Up :

In this unit, we have learnt that Interactive SQL is a utility for entering SQL statements. Interactive SQL (dbisql) is a graphical utility included with SAP Sybase IQ that lets you execute SQL statements, build scripts, and display database data.

We even learnt much about SQL data type and learnt that it is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL. SQL Server offers six categories of data types for use; they are Exact Numeric Data Types, Approximate Numeric Data Types, Date and Time Data Types, Character Strings Data Types, Unicode Character Strings Data Types, Binary Data Types.

6.4 Answers for Check Your Progress :

- **Check Your Progress 1 :**

  1 : a

6.5 Glossary :

1. **DDL :** Data Definition Language
2. **SQL :** Structured Query Language

6.6 Assignment :

Discuss the various data types in SQL.

6.7 Activities :

What do you understand by Interactive SQL ?

6.8 Case Study :

Explain the interactive SQL command file.

6.9 Further Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw – Hill Publication.
3. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
4. Database Management System by Raghu Ramkrishnan – Tata McGraw – Hill Publication.
5. SQL, PL/SQL : The Programming Language Oracle – Ivan Bayross – BPB Publication.

Unit 07 ORACLE FUNCTIONS

UNIT STRUCTURE

7.0 Learning Objectives :

After learning this unit, you will be able to understand :

- The various functions of Oracle.
- The various library functions in Oracle.
- How to use the various SQL functions ?
- The format of various SQL functions.

7.1 Introduction :

Oracle functions operate on —appropriate data|| to transform a value to another value. For example, using a simple calculator, we commonly use the square root function to compute the square root of some number. In this case, the square root key on the calculator calls the square root function and the number in the display is transformed into its square root value. In the square root case, —appropriate data|| is a positive number. For the sake of defining the scope of this discussion, we also consider the square root key on a calculator as a one–to–one function. By one–to–one we mean that if one positive number is furnished, then one square root results from pressing the square root key – a one–to one transformation SQRT, the resulting number as —Answer,|| the equal sign as meaning —is assigned to,|| and the number to be operated on as —original_value,|| then the function could be written like this:

Answer = SQRT (original_value)

Where original_value is a positive number. In alge?ra, the allowable values of original_value are called the domain of the function, which in this case is the set of non–negative numbers. Answer is called the range of the function. Original_value in this example is called the argument of the function SQRT. Often times in computer situations, there is also an upper limit on the domain and range, but theoretically; there is no upper limit in alge?ra. The lower limit on the domain is zero as the square root of negative numbers is undefined unless one ventures into the area of complex numbers, which is beyond the scope of this discussion. Almost any programming language uses functions similar to those found on calculators. In fact, most programming languages go far beyond the calculator functions. Oracle's SQL contains a rich variety of functions. We can categorize Oracle's SQL functions into simple SQL functions, numeric functions, statistical functions, string functions, and date functions. In this chapter, we selectively illustrate several functions in each of these categories. We start by discussing simple SQL functions.

7.2 Library Functions :

Library functions and procedures are utility operations that you can add to any service, physical, logical, or library. Library functions and procedures:

- Have a kind of library
- Are not marked as primary or non–primary
- Have a visibility of public, protected or private

A library function can return values, but has no side effects. A library procedure can return values and can have side effects.

You can declare a library function or procedure visually in Workshop for WebLogic, but you must still write the function body in the Source tab using XQuery. Alternatively, you can write the entire function or procedure and its pragma statement in XQuery.

You can call a library function or procedure from the service, the dataspace project, or from a client application, depending on the visibility level you set.

❑ **Check Your Progress – 1 :**

1. ________ functions and procedures are utility operations that you can add to any service, physical, logical, or library.

   a. Library   b. Oracle

2. A library function ________ return values, but has no side effects.

   a. Can   b. Cannot

7.3 Use of SQL Functions :

Oracle's SQL contains a rich variety of functions. We can categorize Oracle's SQL functions into simple SQL functions, numeric functions, statistical functions, string functions, and date functions. In this portion, we selectively illustrate several functions in each of these categories. We start by discussing simple SQL functions.

1. Simple SQL Functions :

Oracle has a large number of simple functions. Wherever a value is used directly or computed in a SQL statement, a simple SQL function may be used. To illustrate the above square root function, suppose that a table named Measurement contained a series of numeric measured values like this :

Subject	Value
First	35.78
Second	22.22
Third	55.55

We could display the table with this SQL query :

```
SELECT *
FROM measurement
```

Note : We will not use semicolons at the end of SQL statement illustrations; to run these statements in Oracle from the command line, a semicolon must be added. From the editor, a slash (/) is added to execute the statement and no semicolon is used.

We could also generate the same result set with this SQL query :

```
SELECT subject, value
FROM measurement
```

Using the latter query, and adding a square root function to the result set, the SQL query would look like this:

```
SELECT subject, value, SQRT(value)
FROM measurement
```

This would give the following result :

SUBJECT	VALUE	SQRT(VALUE)
First	35.78	5.98163857
Second	22.22	4.7138095
Third	55.55	7.45318724

2. Numeric Functions :

In this section we present and discuss several useful numeric functions, which we divide into the following categories: common numerical manipulation functions, near value functions, null value functions, log and exponential functions, ordinary trigonometry functions, and hyperbolic trignometrical functions.

Common Numerical Manipulation Functions

These are functions that are commonly used in numerical manipulations. Examples of common numerical manipulation functions include:

ABS – Returns the absolute value of a number orvalue.

SQRT – Returns the square root of a number orvalue.

MOD – Returns the remainder of n/m where bothn and m are integers.

SIGN – Returns 1 if the argument is positive; –1 ifthe argument is negative; and 0 if the argument isnegative

Next we present a discussion on the use of these common numerical manipulation functions. Suppose we had a table that looked like this:

```
DESC function_illustrator
```

Which would give :

Name	Null ?	Type
LINENO		NUMBER(2)
VALUE		NUMBER(6,2)

Now, if we typed :

```
SELECT *
FROM function_illustrator
ORDER BY lineno
```

We would get :

LINENO	VALUE
0	9
1	3.44
2	3.88
3	–6.27
4	–6.82
5	0
6	2.5

Now, suppose we use our functions to illustrate the transformation for each value of VALUE :

```
SELECT lineno, value, ABS(value), SIGN(value), MOD(lineno,3)
FROM function_illustrator
ORDER BY lineno
```

We would get :

LINENO	VALUE	ABS(VALUE) SIGN(VALUE)	MOD(LINENO,3)
0	9	9	1 0
1	3.44	3.44	1 1
2	3.88	3.88	1 2
3	–6.27	6.27	–1 0
4	–6.82	6.82	–1 1
5	0	0	0 2
6	2.5	2.5	1 0

Notice the ABS returns the absolute value of VALUE.SIGN tells us whether the value is positive, negative, or zero. MOD gives us the remainder of LINENO/3.All of the common numerical functions take one argument except MOD, which requires two.

Had we tried to include SQRT in this example ourquery would look like this :

```
SELECT lineno, value, ABS(value), SQRT(value), SIGN(value),
```

MOD(lineno,2)

FROM function_illustrator

This would give us:

ERROR:

ORA–01428: argument ‗–6.27' is out of range

no rows selected

In this case, the problem is that there are negative numbers in the value field and SQRT will not accept such values in its domain. Functions can be nested; we can have a function operate on the value produced by another function. To illustrate a nested function we can use the ABS function to ensure that the SQRT function sees only a positive domain. The following query handles both positive and negative numbers:

SELECT lineno, value, ABS(value), SQRT(ABS(value))

FROM function_illustrator

ORDER BY lineno

This would give us :

LINENO	VALUE	ABS(VALUE)	SQRT(ABS(VALUE))
0	9	9	3
1	3.44	3.44	1.8547237
2	3.88	3.88	1.96977156
3	–6.27	6.27	2.50399681
4	–6.82	6.82	2.61151297
5	0	0	0
6	2.5	2.5	1.5811388

3. Statistical Functions :

Near Value Functions

These are functions that produce values near what you are looking for.

Examples of near value functions include:

CEIL – Returns the ceiling value (next highest integer above a number).

FLOOR – Returns the floor value (next lowest integer below number).

TRUNC – Returns the truncated value (removes decimal part of a number, precision adjustable).

ROUND – Returns the number rounded to nearest value (precision adjustable).

Next we present illustrations and a discussion on the use of these near value functions. The near value functions will round off a value in different ways. To illustrate with the data in Function_illustrator, consider this query:

SELECT lineno, value, ROUND(value), TRUNC(value), CEIL(value), FLOOR(value)

FROM function_illustrator

You will get :

LINE NO	VALUE	ROUND (VALUE)	TRUNC (VALUE)	CEIL (VALUE)	FLOOR (VALUE)
0	9	9	9	9	9
1	3.44	3	3	4	3
2	3.88	4	3	4	3
3	–6.27	–6	–6	–6	–7
4	–6.82	–7	–6	–6	–7
5	0	0	0	0	0
6	2.5	3	2	3	2

ROUND will convert a decimal value to the next highest absolute value if the value is 0.5 or greater. Note the way the value is handled if the value of VALUE is negative. —Next highest absolute value‖ for negative numbers rounds to the negative value of the appropriate absolute value of the negative number; e.g.,

ROUND(–6.8) = –7.

TRUNC simply removes decimal values.

CEIL returns the next highest integer value regardless of the fraction. In this case, —next highest‖refers to the actual higher number whether positive or negative. FLOOR returns the integer below the number, again regardless of whether positive or negative. The ROUND and TRUNC functions also may have a second argument to handle precision, which here means the distance to the right of the decimal point.

So, the following query:

SELECT lineno, value, ROUND(value,1), TRUNC(value,1)

FROM function_illustrator

Will give :

LINENO	VALUE	ROUND(VALUE,1)	TRUNC(VALUE,1)
0	9	9	9
1	3.44	3.4	3.4
2	3.88	3.9	3.8
3	–6.27	–6.3	–6.2
4	–6.82	–6.8	–6.8
5	0	0	0
6	2.5	2.5	2.5

The value 3.88, when viewed from one place to the right of the decimal point, rounds up to 3.9 and truncates to 3.8. The second argument defaults to 0 as previously illustrated. The following query may be compared with previous versions, which have no second argument:

SELECT lineno, value, ROUND(value,0), TRUNC(value,0)

FROM function_illustrator

Which will give :

LINENO	VALUE	ROUND(VALUE,0)	TRUNC(VALUE,0)
0	9	9	9
1	3.44	3	3
2	3.88	4	3
3	–6.27	–6	–6
4	–6.82	–7	–6
5	0	0	0
6	2.5	3	2

In additio n, the second argument, precision, may be negative, which means displacement to the left of the decimal point, as shown in the following query :

SELECT lineno, value, ROUND(value,–1), TRUNC(value,–1)

FROM function_illustrator

Which will give ?

LINENO	VALUE	ROUND(VALUE,–1)	TRUNC(VALUE,–1)
0	9	10	0
1	3.44	0	0
2	3.88	0	0
3	–6.27	–10	0
4	–6.82	–10	0
5	0	0	0
6	2.5	0	0

In this example, with –1 for the precision argument, values less than 5 will be truncated to 0, and values of 5 or greater will be rounded up to 10.

Null Value Function :

This function is used if there are null values. The nullvalue function is :

NVL–Returns a substitute (some other value) ifa value is null. NVL takes two arguments. The first argument is the field or attribute that you would like to look for the nullvalue in, and the second argument is the value that you want to replace the null value by. For example, in the statement —NVL (value, 10)||, we are looking for null values in the —value|| column, and would like to replacethe null value in the —value|| column by 10. To illustrate the null value function through anexample, let's insert another row into our Function_illustrator table, as follows :

INSERT INTO function_illustrator values (7, NULL)

Now, if you type :

SELECT *

FROM function_illustrator

You will get :

LINENO	VALUE
0	9
1	3.44
2	3.88
3	–6.27
4	–6.82
5	0
6	2.5

Note that line no 7 has a null value. To give a value of 10 to value for line no = 7, type :

SELECT lineno, NVL(value, 10)

From function_illustrator

You will get :

LINENO	NVL(VALUE,10)
0	9
1	3.44
2	3.88
3	–6.27
4	–6.82
5	0
6	2.5
7	10

Note that a value of 10 has been included for line no 7. But NVL does not change the actual data in the table. It only allows you to use some number in place of null in the SELECT statement (for example, if you aredoing some calculations).

Log and Exponential Functions

SQL's log and exponential functions include :

LN–Returns natural logs, that is, logs with respect to base e.

LOG–Returns base 10 log.

EXP–Returns e raised to a value.

POWER–Returns value raised to some exponential ower.

To illustrate these functions, look at the following examples :

Example 1 : Using the LN function :

SELECT LN(value)

FROM function_illustrator

WHERE lineno = 2

This will give :

```
LN(VALUE)
----------------
1.35583515
```

Example 2 : Using the LOG function :

The LOG function requires two arguments. The first argument is the base of the log, and the second argument is the number that you want to take the log of. In the following example, we are taking the log of 2, base value.

```
SELECT LOG(value, 2)
FROM function_illustrator
WHERE lineno = 2
```

This will give:

```
LOG(VALUE,2)
----------------
.511232637
```

As another example, you if want to get the log of 8, base 2, you would type :

```
SELECT LOG(2,8)
FROM function_illustrator
WHERE rownum = 1
```

Giving :

```
LOG(2,8)
----------------
```

Example 3 : Using the EXP function :

```
SELECT EXP(value)
FROM function_illustrator
WHERE lineno = 2
```

Gives :

```
EXP(VALUE)
----------------
48.4242151
```

Example 4 : Using the POWER function :

The POWER function requires two arguments. The first argument is the value that you would like raised to some exponential power, and the second argument is the power (exponent) that you would like the number raised to. See the following example :

```
SELECT POWER(value,2)
FROM function_illustrator
WHERE lineno = 0
```

Which gives:

```
POWER(VALUE,2)
----------------
```

Ordinary Trigonometry Functions

SQL's ordinary trigonometry functions include :

SIN–Returns the sine of a value.

COS–Returns the cosine of a value.

TAN–Returns the tangent of a value.

The SIN, COS, and TAN functions take arguments in radians where,

radians = (angle * 2 * 3.1416 / 360)

To illustrate the use of the ordinary trigonometric functions, let's suppose we have a table called Trig with the following description:

```
DESC trig
```

Will give :

```
Name Null? Type
------------------------------------------------------------
VALUE1 NUMBER(3)
VALUE2 NUMBER(3)
VALUE3 NUMBER(3)
```

14

Common Oracle Functions : A Function Review

And,

```
SELECT *
FROM trig
```

Will give :

```
VALUE1       VALUE2       VALUE3
------------ ------------ ------------
    30           60           90
```

Example 1 : Using the SIN function to find the sine of 30 degrees :

```
SELECT SIN(value1*2*3.1416/360)
FROM trig
```

Gives :

```
SIN(VALUE1*2*3.1416/360)
---------------------
.50000106
```

Example 2 : Using the COS function to find the cosine of 60 degrees :

```
SELECT COS(value2*2*3.1416/360)
FROM trig
```

Gives :

```
COS(VALUE2*2*3.1416/360)
-----------------------------------
.499997879
```

Example 3 : Using the TAN function to find the tangent of 30 degrees :

```
SELECT TAN(value1*2*3.1416/360)
FROM trig
```

Gives :

```
TAN(VALUE1*2*3.1416/360)
-----------------------------------
.577351902
```

Hyperbolic Trig Functions

SQL's hyperbolic trigonometric functions include: SINH–Returns the hyperbolic sine of a value. COSH–Returns the hyperbolic cosine of a value. TANH–Returns the hyperbolic tangent of a value.

These hyperbolic trigonometric functions also take arguments in radians where, radians = (angle * 2 * 3.1416 / 360)

We illustrate the use of these hyperbolic functions with examples :

Example 1 : Using the SINH function to find the hyperbolic sine of 30 degrees:

```
SELECT SINH(value1*2*3.1416/360)
FROM trig
```

Gives :

```
SINH(VALUE1*2*3.1416/360)
-----------------------------------
.54785487
```

Example 2 : Using the COSH function to find the hyperbolic cosine of 30 degrees :

```
SELECT COSH(value1*2*3.1416/360)
FROM trig
```

Gives :

```
COSH(VALUE1*2*3.1416/360)
-----------------------------------
1.14023899
```

Example 3 : Using the TANH function to find the hyperbolic tangent of 30 degrees :

```
SELECT TANH(value1*2*3.1416/360)
FROM trig
```

Gives :

```
TANH(VALUE1*2*3.1416/360)
-----------------------------------
.48047372
```

In terms of usage, the common numerical manipulation functions (ABS, MOD, SIGN, SQRT), the ―near value‖ functions (CEIL, FLOOR, ROUND, TRUNC), andNVL (an Oracle exclusive null handling function) are used often. An engineer or scientist might use the LOG, POWER, and trig functions.

4. Aggregate Functions :

SQL> SELECT * FROM TEAMSTATS;

➢ **OUTPUT :**

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

6 rows selected.

➢ **COUNT :**

COUNT the number of rows that satisfy the condition in the WHERE clause. how many ball players were hitting under 350.

➢ **INPUT/OUTPUT :**

SQL> SELECT COUNT(*) FROM TEAMSTATS WHERE HITS/AB < .35;

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

➢ **SUM :**

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

SQL> SELECT SUM(SINGLES) TOTAL_SINGLES FROM TEAMSTATS;

```
TOTAL_SINGLES
-------------
          174
```

SQL> SELECT SUM(SINGLES) TOTAL_SINGLES, SUM(DOUBLES) TOTAL_DOUBLES,

SUM(TRIPLES) TOTAL_TRIPLES, SUM(HR) TOTAL_HR FROM TEAMSTATS;

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

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

SQL> SELECT SUM(SINGLES) TOTAL_SINGLES, SUM(DOUBLES) TOTAL_DOUBLES,

SUM(TRIPLES) TOTAL_TRIPLES, SUM(HR) TOTAL_HR FROM TEAMSTATS WHERE HITS/AB >=300;

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

To compute a team batting average, type

SQL> SELECT SUM(HITS)/SUM(AB) TEAM_AVERAGE FROM TEAMSTATS;

```
TEAM_AVERAGE
------------
   .33706294
```

➢ **AVG :**

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

SQL> SELECT AVG(SO) AVE_STRIKE_OUTS FROM TEAMSTATS;

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

The following example illustrates the difference between SUM and AVG :

➢ **INPUT/OUTPUT :**

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

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

➢ **MAX :**

what is the highest number of hits?

SQL> SELECT MAX(HITS) FROM TEAMSTATS;

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

➢ **MIN :**

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

SQL> SELECT MIN(AB) FROM TEAMSTATS;

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

➢ **INPUT/OUTPUT :**

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

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

➢ **INPUT/OUTPUT :**

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

```
COUNT(AB) AVG(AB) MIN(AB) MAX(AB) STDDEV(AB) VARIANCE(AB) SUM(AB)
------  -----  -----  -----  ------    ------    ------
7       119.167    1    187  75.589    5712.97      715
```

5. String Functions :

A host of string functions are available in Oracle. String functions refer to alphanumeric charactertrings. Among the most common string functions are INSTR, SUBSTR, REPLACE, and TRIM. Here wepresent and discuss these string functions. INSTR, SUBSTR, and REPLACE,‖ Regular Expressions : String Searching and Oracle10g.‖

The INSTR Function the INSTR Function INSTR (—in–string‖) is a function used to find patternsin strings. By patterns we mean a series of alphanumeric characters. The general syntax of INSTR is : INSTR (string to search, search pattern [, start [,occurrence]]) The arguments within brackets ([]) are optional. We will illustrate each argument with examples. INSTR returns a location within the string where search patternegins. Here are some examples of the use of the INSTR function :

SELECT INSTR(‗This is a test','is')

FROM dual

This will give :

INSTR(‗THISISATEST','IS')

Common Oracle Functions : A Function Review The first character of string to search is numbered 1. Since —is‖ is the search pattern, it is found in string to search at position 3. If we had chosen to look for the second occurrence of —is,‖ the query would look like this:

SELECT INSTR(‗This is a test','is',1,2)

FROM dual

And the result would be :

INSTR(‗THISISATEST','IS',1,2)

In this case, the second occurrence of —is‖ is found at position 6 of the string. To find the second occurrence, we have to tell the function where to start; therefore the third argument starts the search in position 1 of string to search. If a fourth argument is desired, then the third argument is mandatory. If search pattern is not in the string, the INSTR function returns 0, as shown by the query below :

```
SELECT INSTR(‗This is a test','abc',1,2)
FROM dual
Which would give :
INSTR(‗THISISATEST','ABC',1,2)
-----------------------------
```

The SUBSTR function returns part of a string. The general syntax of the function is as follows :

```
SUBSTR(original string, begin [,how far])
```

An original string is to be dissected beginning at the begin character. If no how far amount is specified, then the rest of the string from the begin point is retrieved. If begin is negative, then retrieval occurs from the right–hand side of original string. Below is an example:

```
SELECT SUBSTR(‗My address is 123 Fourth St.',1,12)
FROM dual
Which would give :
SUBSTR(‗MYAD)
-------------
My address i
```

Here, the first 12 characters are returned from original string. The first 12 characters are specified since begin is 1 and how far is 12. Notice that blanks count as characters. Look at the following query :

```
SELECT SUBSTR(‗My address is 123 Fourth St.',5,12)
From dual
This would give :
SUBSTR(‗MYAD)
-------------
address is 12
```

In this case, the retrieval begins at position 5 and again goes for 12 characters.

Here is an example of retrieval with no third argument, meaning it starts at begin and retrieves therest of the string :

```
SELECT SUBSTR (‗My address is 123 Fourth St.',6)
FROM dual
This would give :
SUBSTR(‗MYADDRESSIS123F
---------------------
dress is 123 Fourth St.
```

SUBSTR may also retrieve from the right–hand side of original string, as shown below :

SELECT SUBSTR(‗My address is 123 Fourth St.',–9,5)

FROM dual

This would give :

SUBST

----

ourth

The result comes from starting at the right end of the string and counting backward for nine characters, then retrieving five characters from that point. Often in string handling, SUBSTR and INSTR areused together. For example, if we had a series ofnames in last name, first name format, e.g., ―Harrison, John Edward,‖ and wanted to retrieve first and middlenames, we could use the commaand space to find the end of the last name. This is particularly useful sincethe last name is of unknown length and we rely only onthe format of the names for retrieval, as shown below :

SELECT SUBSTR(‗Harrison, John Edward', INSTR(‗Harrison, John Edward',', ?)+2)

FROM dual

This would give :

SUBSTR(‗HAR

John Edward

The original string is ―Harrison, John Edward.‖ The begin number has been replaced by the INSTR function, which returns the position of the command blank space. Since INSTR is using two characters to find the place to begin retrieval, the actual retriev almust begin two characters to the right of that point. If we do not move over two spaces, and then we get this :

SELECT SUBSTR(‗Harrison, John Edward', INSTR(‗Harrison, John Edward',', ‗))

FROM dual

This would give :

SUBSTR(‗HARRI

, John Edward

The result includes the command space because retrieval starts where the INSTR function indicated the position of search pattern occurred. If the INSTR pattern is not found, then the entirestring would be returned, as shown by this query : SELECT SUBSTR(‗Harrison, John Edward', INSTR(‗Harrison, John Edward','zonk'))

FROM dual

This would give :

SUBSTR(‗HARRISON,JOHN

Harrison, John Edward

which is actually this :

SELECT SUBSTR(‗Harrison, John Edward',0)

FROM dual

which would give :

SUBSTR(‗HARRISON,JOHN

Harrison, John Edward

The REPLACE Function is a common situation to not only find apattern (INSTR) and perhaps extract it (SUBSTR), but then to replace the value(s) found. The REPLACE function has the following general syntax:

REPLACE (string, look for, replace with) where all three arguments are necessary. The look forstring will be replaced with the replace with stringe very time it occurs.

Here is an example :

SELECT REPLACE (‗This is a test',' is ‗,' may be ‗)

FROM dual

This gives :

REPLACE(‗THISISATE

This may be a test

Here the look for string consists of ―is‖, including the spaces before and after the word ―is.‖ It does not matter if the look for and the replace with strings are of different lengths. If the spaces are not placed around

―is‖, then the ―is‖ in ―This‖ will be replaced along with the word ―is‖, as shown by the following query :

SELECT REPLACE (‗This is a test','is',' may be ‗)

FROM dual

This would give :

REPLACE(‗THISISATEST','IS'

This may be a test

If the look for string is not present, then the replacing does not occur, as shown by the following query :

SELECT REPLACE (‗This is a test','glurg',' may be ‗)

FROM dual

Which would give :

```
REPLACE(‗THISI
--------------
This is a test
```

The TRIM Function the TRIM Function TRIM is a function that removes characters from theleft or right ends of a string or both ends. The TRIM function was added in Oracle 9. Originally, LTRIM andRTRIM were used for trimming characters from the left or right ends of strings. TRIM supercedes both ofthese. The general syntax of TRIM is :

TRIM ([where] [trim character] FROM subject string)

The optional where is one of the keywords

—leading,‖ —trailing,‖ or —both.‖

If the optional trim character is not present, then blanks will be trimmed. Trim character may be any character. The word FROM is necessary only if where or trim character is present. Here is an example:

```
SELECT TRIM (‗This string has leading and trailing Spaces')
FROM dual
```

Which gives :

```
TRIM(‗THISSTRINGHASLEADINGANDTRAILINGSPACES
-----------------------------------------------
This string has leading and trailing spaces
```

Both the leading and trailing spaces are deleted. This is probably the most common use of the function. We canbe more explicit in the use of the function, as shown in the following query :

```
SELECT TRIM (both ‗ ‗ from ‗ String with blanks ‗)
FROM dual
```

Which gives :

```
TRIM(BOTH''FROM'ST
---------------
String with blanks
```

In these examples, characters rather than spaces are trimmed :

```
SELECT TRIM(‗F' from ‗Frogs prefer deep water')
FROM dual
```

Which would give :

```
TRIM(‗F'FROM'FROGSPREF
--------------------
rogs prefer deep water
```

Here are some other examples.

Example 1 :

```
SELECT TRIM(leading ‗F' from ‗Frogs prefer deep water')
FROM dual
```

Which would give :

```
TRIM(LEADING'F'FROM'FR
----------------------
rogs prefer deep water
```

Example 2 :

SELECT TRIM(trailing ‗r' from ‗Frogs prefer deep water')
FROM dual

Which would give :

```
TRIM(TRAILING'R'FROM'F
----------------------
Frogs prefer deep wate
```

Example 3 :

SELECT TRIM (both ‗z' from ‗zzzzz I am asleep zzzzzz')
FROM dual

Which would give :

```
TRIM(BOTH'Z'F
--------------
I am asleep
```

In the last example, note that the blank space was preserved because it was not trimmed. To get rid of the leading/trailing blank(s) we can nest TRIMs like this :

SELECT TRIM(TRIM (both ‗z' from ‗zzzzz I am asleep zzzzzz'))
FROM dual

This would give :

```
TRIM(TRIM(B
___________
I am asleep
```

5. Date functions :

Oracle's date functions allow one to manage and handle dates in a far easier manner than if one had to actually create calendar tables or use complex algorithms fordate calculations. First we must note that the date data type is not a character format. Columns with date data types contain both date and time. We must format dates to see all of the information contained in a date.

If you type :

SELECT SYSDATE
FROM dual

You will get :

```
SYSDATE
----------
10–SEP–06
```

The format of the TO_CHAR function (i.e., convert to acharacter string) is full of possibilities. Here is anexample:

SELECT TO_CHAR(SYSDATE, ‗dd Mon, yyyy hh24:mi:ss')

FROM dual

This gives:

TO_CHAR(SYSDATE,'DDMO

----

10 Sep, 2006 14:04:59

This presentation gives us not only the date in ―dd Monyyyy‖ format, but also gives us the time in 24–hours, minutes, and seconds.

We can add months to any date with the ADD_

MONTHS function like this :

SELECT TO_CHAR(SYSDATE, ‗ddMONyyyy') Today,

TO_CHAR(ADD_MONTHS(SYSDATE, 3), ‗ddMONyyyy') ―+ 3 mon‖,

TO_CHAR(ADD_MONTHS(SYSDATE, –23), ‗ddMONyyyy') ― -23 mon‖

FROM dual

This will give us :

TODAY + 3 mon – 23 mon

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

10SEP2006 10DEC2006 10OCT2004

In this example, note that the ADD_MONTHS function is applied to SYSDATE, a date data type, and then the result is converted to a character string with

TO_CHAR.

The LAST_DAY function returns the last day of any month, as shown in the following query:

SELECT TO_CHAR(LAST_DAY(‗23SEP2006'))

FROM dual

This gives us :

TO_CHAR(L

30–SEP–06

This example illustrates that Oracle will convert characterdates to date data types implicitly. There is also a TO_DATE function to convert from characters to dates explicitly. It is usually not a good idea to take advantage of implicit conversion, and therefore a more proper version of the above query would look like this:

SELECT TO_CHAR(LAST_DAY(TO_DATE(‗23SEP2006','ddMON yyyy')))

FROM dual

This would give us :

```
TO_CHAR(L
----------
30–SEP–06
```

In the following example, we convert the date'23SEP2006' to a date data type, perform a date functionon it (LAST_DAY), and then reconvert it to a character data type. We can change the original date format in the TO_CHAR function as well, as shown below:

```
SELECT TO_CHAR(LAST_DAY(TO_DATE(‗23SEP2006','ddMON yyyy')),
‗Month dd, yyyy')
FROM dual
```

This will give us :

```
TO_CHAR(LAST_DAY(T
-------------------
September 30, 2006
```

To find the time difference between two dates, use the MONTHS_BETWEEN function, which returns fractional months. The general format of the function is:

```
MONTHS_BETWEEN(date1, date2)
```

where the result will be date1 – date2.

Here is an example:

```
SELECT MONTHS_BETWEEN(TO_DATE(‗22SEP2006','ddMONyyyy'),
TO_DATE(‗13OCT2001','ddMONyyyy')) ―Months difference‖
FROM dual
```

This gives :

```
Months difference
----------------
59.2903226
```

Here we explicitly converted our character string dates to date data types before applying the MONTHS_BETWEEN function.

The NEXT_DAY function tells us the date of the day of the week following a particular date, where ―dayof the week‖ is expressed as the day written out (like Monday, Tuesday, etc.):

```
SELECT NEXT_DAY(TO_DATE(‗15SEP2006','DDMONYYYY'), 'Monday')
FROM dual
```

This gives :

```
NEXT_DAY(
----------
18–SEP–06
```

The Monday after 15–SEP–06 is 18–SEP–06, which is displayed in the default date format.

❑ **Check Your Progress – 2 :**

1. Wherever a value is used directly or computed in a SQL statement, a simple SQL ________ may be used.

   a. function   b. Relation

2. ________ will convert a decimal value to the next highest absolute value.

   a. ROUND   b. ROUNDOFF

3. ________ returns the next highest integer value regardless of the fraction.

   a. CEIL   b. RETURN

4. The ________ Function is a function used to find patterns in strings.

   a. INSTR   b. CEIL

7.4 Let Us Sum Up :

In this unit, we have learnt that Library functions and procedures are utility operations that you can add to any service, physical, logical, or library. A library function can return values, but has no side effects. A library procedure can return values and can have side effects.

Oracle's SQL contains a rich variety of functions. We can categorize Oracle's SQL functions into simple SQL functions, numeric functions, statistical functions, string functions, and date functions.

7.5 Answers for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a   **2 :** a

❑ **Check Your Progress 2 :**

1 : a   **2 :** a   **3 :** a   **4 :** a

7.6 Glossary :

1. **SQL :** Structured Query Language.

2. **Oracle :** It is an object–relational database management system [3] produced and marketed by Oracle Corporation.

7.7 Assignment :

Discuss the utility of Oracle SQL functions.

7.8 Activities :

Discuss the numeric functions of Oracle with the help of an illustration.

7.9 Case Study :

With the help of an example discuss the string function in oracle.

7.10 Further Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw – Hill Publication.
3. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
4. Database Management System by Raghu Ram Krishnan – Tata McGraw – Hill Publication.
5. SQL, PL/SQL: The Programming Language Oracle – Ivan Bayross – BPB Publication.

BLOCK SUMMARY :

In this block we learnt about the basic concepts of query writing and sub queries. In this block we learnt abou the create statement, apart from this the writing queries has even been learnt by in very detail in detail. On the other hand we even learnt about the table and view option. The decode statement, insert, update and delete commands have been discussed in detail with sufficient examples.

The interactive SQL portion was even been discussed here under this block. The author has explained the various data types briefly. Apart from this the objective of this block includes detailing the studentsabout the various Oracle functions. Here under oracle functions the topics covered are the library functions as well as the uses of various SQL functions in a very detail with the help of sufficient and suitable examples.

BLOCK ASSIGNMENT :

❖ **Short Questions :**

1. Oracle
2. SQL
3. Library functions
4. Update Command
5. Interactive SQL

❖ **Long Questions :**

1 Explain the use of Insert command in Oracle SQL with the help of an example.

2 Explain the various components of Update command.

3 Discuss the Delete command in SQL.

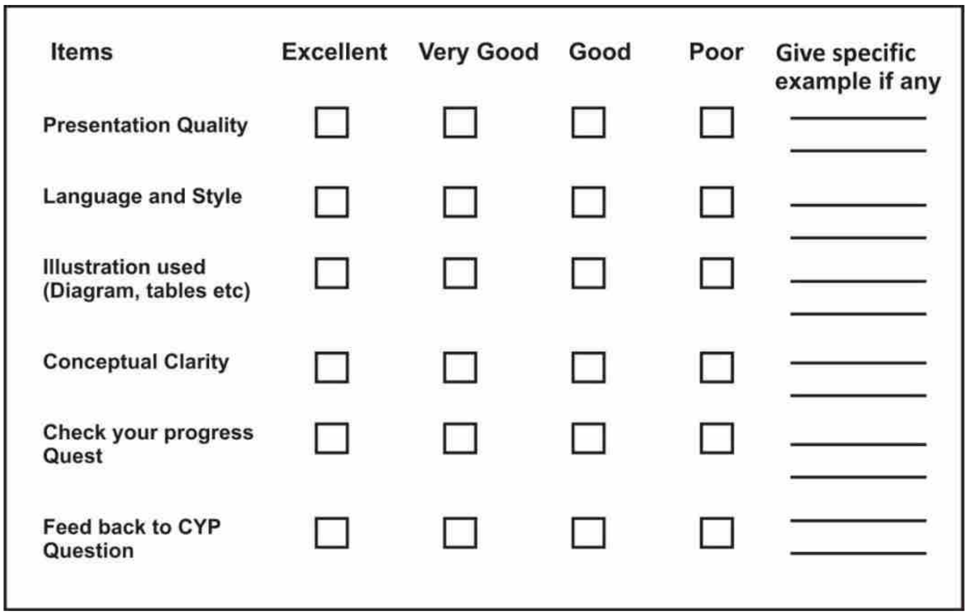
4 Discuss the date function in oracle with the help of a suitable function.

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	4	5	6	7
No. of Hrs.				

2. Please give your reactions to the following items based on your reading of the block :



Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	
Language and Style	☐	☐	☐	☐	
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	
Conceptual Clarity	☐	☐	☐	☐	
Check your progress Quest	☐	☐	☐	☐	
Feed back to CYP Question	☐	☐	☐	☐	

3. Any other Comments

....

....

....

....

....

....

....

....

Dr. Babasaheb Ambedkar Open University Ahmedabad

BCAR 302

RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)

BLOCK 3 : PL/SQL–I

PL/SQL–I

Block Introduction :

This block discusses constructs of PL/SQL that enable software developers to use this procedural language interface to the Oracle RDBMS. Constructs within PL/SQL are similar to those found in a 3GL and C and provide a flexible method to manipulate database information. PL/SQL is a mainstay of SQL* Forms, but writing database procedures outside forms gives you access to common database manipulation routines by other Oracle development tools while limiting the size of forms applications.

The aim of this block is to ena?le the reader understand the basic concepts of PL/SQL and their role. The writer has tried his best to introduce the PL/SQL with its features and advantages, apart from this the variables and constants have been discuss in detail. The reader will also get a detailed insight of data types and arrays. On the other hand the writer has even discussed the declare statement, apart from this various PL/SQL illustrations and examples have been provided in this block.

In this block, we will learn about Store procedure, Triggers. It provides an overview of the above topic with the help of various examples.

You will also get a basic idea of what store procedure. Various functions and commands have been discussed here in this block in a very brief and interesting way. The main topics that have been covered in these block are stored procedures, internal functions and database triggers.

The writer has tried his best to concise the topics using the easiest language to help the student to understand the topic.

The aim of this block is to enable the reader understand the basic concepts of these topcis. The writer has tried his best to detail the store procedure; apart from this the triggers have even been discussed by him in detail sufficient examples.

The objective of this block includes detailing the student's about the various above mentioned topics in very detail with the help of sufficient and suitable examples.

Block Objectives :

After learning this block, you will be able to understand :

- Discuss about PL/SQL
- About the Variables and Constants
- About various Data types and Arrays
- Control Statements
- DECLARE statement and Naming Conventions
- Purpose of stored procedures and their creation
- How to delete and replace a stored procedure
- Procedure overloading
- The triggers and their use
- The process of creation of triggers
- The rule of firing of triggers
- The Package Creation
- Utility of Package Subprograms

Block Structure :

Unit

08

INTRODUCTION TO PL/SQL

UNIT STRUCTURE

8.0 Learning Objectives :

After learning this unit, you will be able to understand :

- The Introduction to PL/SQL
- How to describe PL/SQL Architecture
- The Declares Variables

8.1 Introduction :

Oracle's PL/SQL is a procedural language extension of SQL. It is the standard programming language for Oracle RDBMS and follows the procedural approach. It also provides conditional constructs like IF. THEN, WHILE LOOP which is available in other languages like C, Pascal, COBOL. PL/SQL is very powerful as it combines the flexibility available in SQL with the procedural constructs. Oracle's PL/SQL adds a lot of additional capabilities to Oracle programming in order to do validations, customization, include user interfaces and handling errors effectively.

➢ **What is PL/SQL ?**

PL/SQL is a block–structured language. The basic unit of PL/SQL is called a BLOCK, which contains declarative statements, executable statements and error handling statements. These blocks can be nested into one or more blocks.

➢ **Features of PL/SQL :**

- Allows users to embed one or more SQL statements together and execute as asingle SQL unit.

- Allows declaration of variables.
- Allows usage of conditional constructs
- Allows programming of error–handling using exceptions
- Allows row–by–Row processing of data using cursors
- Allows triggers to be created and fired.

8.3 Advantages of PL/SQL :

- **SQL Support :**

SQL has become the standard database language because it is flexible, powerful and easy to learn. Since it is a non–procedural language, a user need not know how the statements are processed but just needs to indicate the requirement.

PL/SQL allows the usage of all kinds of SQL data manipulation, cursor control and transaction control statements as well as all the SQL functions, operators and pseudo columns.

- **Better Performance :**

Without PL/SQL Oracle would process SQL statements one at a time. Each SQL statement results in another Oracle call to Oracle and higher performance overhead. Since PL/SQL can contain SQL statements, all the SQL statements can be placed inside PL/SQL thereby reducing the time taken for communication between the server and the application. This reduces network traffic and results in better performance.

- **Support for Object Oriented Programming :**

Oracle implements the concept of object oriented programming. This allows the creation of software components that are modular, maintainable and reusable. Using encapsulation operations with the data object types lets users to move data–maintenance codes out of SQL scripts and PL/SQL blocks into methods.

- **Portability :**

Applications written in PL/SQL provide portability to the operating system and platform on which they runs. These applications can run wherever Oracle can run.

- **Higher Productivity :**

PL/SQL adds functionality to Oracle's non–procedural tools like Forms and Reports. These tools can help to build applications using familiar procedural constructs. PL/SQL does not differ in environments. It works the same way as it works for other built–in tools. Also any scripts written using one tool can be used by another tool.

- **Integration with Oracle :**

Oracle and PL/SQL are based on SQL statements. It supports all the SQL Datatypes. These Datatypes integrate PL/SQL with the Oracle Data dictionary.

❑ **Check Your Progress – 1 :**

1. SQL is a ________ language.

   a. Procedural   b. Non–procedural

2. SQL Stands for ________.

   a. Standard Query Language b. Structured Query Language

8.4 Variable :

A PL/SQL variable can have any SQL data type (such as CHAR, DATE, or NUMBER) or a PL/SQL–only data type (such as BOOLEAN or PLS_INTEGER).

Example 1–2 declares several PL/SQL variables. One has a PL/SQL–only data type;the others have SQL data types.

Example 1–2 PL/SQL Variable Declarations

```
SQL> DECLARE
2 part_numberNUMBER(6); -- SQL data type
3 part_nameVARCHAR2(20); -- SQL data type
4 in_stock BOOLEAN; -- PL/SQL–only data type
5 part_priceNUMBER(6,2); -- SQL data type
6 part_descriptionVARCHAR2(50); -- SQL data type
7 BEGIN
8 NULL;
9 END;
10 /
PL/SQL procedure successfully completed.
SQL>
```

\PL/SQL also lets you declare composite data types, such as nested tables, variable–sizearrays, and records.

- With the assignment operator (:=), as in Example 1–3.
- By selecting (or fetching) database values into it, as in Example 1–4.
- By passing it as an OUT or IN OUT parameter to a supprogram, and then assigning the value inside the supprogram, as in Example 1–5

Example 1–3 Assigning Values to Variables with the Assignment Operator

```
SQL> DECLARE -- You can assign values here
2 wages NUMBER;
3 hours_workedNUMBER := 40;
4 hourly_salaryNUMBER := 22.50;
5 bonus NUMBER := 150;
6 country VARCHAR2(128);
7 counter NUMBER := 0;
8 done BOOLEAN;
9 valid_id BOOLEAN;
10 emp_rec1 employees%ROWTYPE;
11 emp_rec2 employees%ROWTYPE;
12 TYPE commissions IS TABLE OF NUMBER INDEX BY
PLS_INTEGER;
```

```
13 comm_tab commissions;
14
15 BEGIN -- You can assign values here too
16 wages := (hours_worked * hourly_salary) + bonus;
17 country := 'France';
18 country := UPPER('Canada');
19 done := (counter > 100);
20 valid_id := TRUE;
21 emp_rec1.first_name := 'Antonio';
22 emp_rec1.last_name := 'Ortiz';
23 emp_rec1 := emp_rec2;
24 comm_tab(5) := 20000 * 0.15;
25 END;
26 /
PL/SQL procedure successfully completed.
SQL>
```

In Example 1–4, 10% of an employee's salary is selected into the bonus variable. Now you can use the bonus variable in another computation or insert its value into a database table.

Example 1–4 Using SELECT INTO to Assign Values to Variables

```
SQL> DECLARE
2 bonus NUMBER(8,2);
3 emp_idNUMBER(6) := 100;
4 BEGIN
5 SELECT salary * 0.10 INTO bonus
6 FROM employees
7 WHERE employee_id = emp_id;
8 END
9 /
PL/SQL procedure successfully completed. PL / SQL
SQL>
```

Example 1–5 passes the new_sal variable to a supprogram, and the Supprogram updates the variable.

```
2 new_salNUMBER(8,2);
3 emp_idNUMBER(6) := 126;
4
5 PROCEDURE adjust_salary (
6 emp_id NUMBER,
7 sal IN OUT NUMBER
8 ) IS
```

```
emp_jobVARCHAR2(10);
avg_salNUMBER(8,2);
BEGIN
SELECT job_id INTO emp_job
FROM employees
WHERE employee_id = emp_id;
SELECT AVG(salary) INTO avg_sal
FROM employees
WHERE job_id = emp_job;
DBMS_OUTPUT.PUT_LINE ('The average salary for '
|| emp_job
|| ' employees: '
|| TO_CHAR(avg_sal)
);

sal := (sal + avg_sal)/2;

END;

BEGIN
SELECT AVG(salary) INTO new_sal
FROM employees;

DBMS_OUTPUT.PUT_LINE ('The average salary for all employees:'
|| TO_CHAR(new_sal)
);

adjust_salary(emp_id, new_sal);
END;
/
The average salary for all employees: 6461.68
The average salary for ST_CLERK employees: 2785
PL/SQL procedure successfully completed.
SQL>
```

❑ **Check Your Progress – 2 :**

1. A PL/SQL ________ can have any SQL data type.

   a. Variable b. Constant

2. ________ also lets you declare composite data types, such as nested tables, variable–size arrays, and records.

   a. PL/SQL b. Variable

8.5 Constant :

As the name implies a constant is a value used in a PL/SQL Block that remains unchanged throughout the program. A constant is a user–defined literal value. You can declare a constant and use it instead of actual value.

For example : If you want to write a program which will increase the salary of the employees by 25%, you can declare a constant and use it throughout the program. Next time when you want to increase the salary again you can change the value of the constant which will be easier than changing the actual value throughout the program.

constant_name CONSTANT datatype := VALUE;

constant_name is the name of the constant i.e. similar to a variable name.

The word CONSTANT is a reserved word and ensures that the value does not change.

VALUE – It is a value which must be assigned to a constant when it is declared.

You cannot assign a value later.

For example, to declare salary_increase, you can write code as follows:

```
DECLARE
salary_increase CONSTANT number (3) := 10;
```

You must assign a value to a constant at the time you declare it. If you do not assign a value to a constant while declaring it and try to assign a value in the execution section, you will get a error. If you execute the below Pl/SQL block you will get error.

```
DECLARE
salary_increase CONSTANT number(3);
BEGIN
salary_increase := 100;
dbms_output.put_line (salary_increase);
END;
```

To declare a constant, put the keyword CONSTANT before the type specifier. The following declaration names a constant of type REAL and assigns an unchangeable value of 5000 to the constant. A constant must be initialized in its declaration. Constants are initialized every time a block or supprogram is entered.

Example 2–7 Declaring Constants

```
SQL> DECLARE
2. credit_limit CONSTANT REAL := 5000.00;
3. max_days_in_year CONSTANT INTEGER := 366;
4. urban_legend CONSTANT BOOLEAN := FALSE;
5. BEGIN
6. NULL;
7. END;
8. /
```

PL/SQL procedure successfully completed.

SQL>

❑ **Check Your Progress – 3 :**

1. ________ is a value used in a PL/SQL Block that remains unchanged through out the program.

   a. Constant b. Variable

2. To declare a constant, put the keyword ________ before the type specifier.

   a. CONSTANT b. DECLARE CONSTANT

8.6 Datatypes & Array :

➢ **Datatypes :**

Each literal or column value manipulated by Oracle has a datatype. A value's datatype associates a fixed set of properties with the value. Broadly classifying the datatypes, they can be of two types :

- BUILT–IN
- USER–DEFINED (dealt in a later chapter)

Built–in datatypes are predefined set of datatypes set in Oracle. Based on the type of data that can be stored, built–in datatypes can be classified as

- Character Datatypes
- Numeric Datatype
- Date Datatype
- Raw Datatype
- Long Raw Datatype
- Lob Datatype
- Character Datatypes Char(n)

Char datatype is a fixed length character data of length n bytes. Default size is 1 byte and it can hold a maximum of 2000 bytes. Character datatypes pad blank spaces to the fixed length if the user enters a value lesser than the specified length.

Syntax

Char(n)

Example:

X char (4) stores upto 4 characters of data in the column X.

Varchar2(size)

Varchar 2 datatypes are variable length character strings. They can store alpha–numeric values and the size must be specified. The maximum length of varchar2 datatype is 4000 bytes. Unlike char datatype, blank spaces are not padded to the length of the string. So, this is more preferred than character datatypes since it does not store the maximum length.

Syntax

Varchar2(Size)

Example:

X varchar2 (10) stores upto 10 characters of data in the column X.

Numeric datatypes

Number

The number datatypes can store numeric values where p stands for the precision and s stands for the scale. The precision can range between 1 to 38 and the scale ranges from –84 to 127.

Syntax

Number (p, s)

Example :

Sal number – Here the scale is 0 and the precision is 38

Sal number(7) – Here the scale is 0 and the number is a fixed point number of 7 digits

Sal number(7,2) – Stores 5 digits followed by 2 decimal points.

➢ **DATE datatype :**

Date datatype is used to store date and time values. The default format is 'DD–MON–YY'. The valid data for a date datatype ranges from January 1, 4712 BC to December 31, 4712 AD. Date datatype stores 7 bytes one each for century, year, month, day, hour, minute and second.

RAW Datatype

RAW(n)

RAW datatype stores binary data of length n bytes. The maximum size is 255 bytes. Specifying the size is a must for this datatype.

Syntax

Raw(n)

LONG Datatype

Stores character data of variable length upto 2 Gigabytes(GB) or 231 – 1.

➢ **LONG RAW Datatype :**

Stores upto 2 Gigabytes(GB) of raw binary data. The use of LONG Values are restricted. The restrictions are :

- A Table cannot contain more than one LONG column
- LONG columns cannot appear in Integrity constraints (dealt later)
- They cannot appear in WHERE, ORDER BY clauses of SELECT statements
- Cannot be a part of expressions or conditions
- Cannot appear in the SELECT list of CREATE TABLE as SELECT statement.

➢ **LOB Datatypes :**

In addition to the above datatypes, Oracle 8 supports LOB datatypes. LOB is the acronym for LARGE OBJECTS. The LOB datatypes stores upto 4 GB of data. This datatype is used for storing video clippings, large images, history documents etc. LOB datatypes can be

- CLOB Character Large Objects (Internal LOB)
- BLOB Binary Large Objects (Internal LOB)
- BFILE Binary File (External LOB)

The LOB datatypes store values, which are called locators. These locators indicate the place where the objects are stored. The location can be out–of–line or in an external file like CDROM.

In order to manipulate the LOB type of data, DBMS_LOB package is used.

➢ **Array :**

An array is an ordered set of data elements. All elements of a given array are of the same datatype. Each element has an index, which is a number corresponding to the element's position in the array. The number of elements in an array is the size of the array. Oracle arrays are of variable size, which is why they are called VARRAYs. The maximum size must be specified while creating varrays.

Declaring a Varray does not occupy space. It defines a type, which can be used as

- The datatype of a column of a relational table.
- An object type attribute.
- A PL/SQL variable, parameter, or function return type.

An array object is stored in line, that is, in the same tablespace as the other data in its row.

The following example illustrates this concept.

Example

```
CREATE OR REPLACE TYPE price_list AS VARRAY(5) OF number(9);
CREATE TABLE prods
(
pno number,
rateprice_list
) ;
```

Inserting Values

Inserting records are always done only using Constructor method.

```
INSERT INTO prods VALUES
(1, price_list (12,34,56,78));
INSERT INTO prods VALUES
(2, price_list (12,34,56));
```

Here the maximum values that can be specified are only 5. The number of values that the column can hold must be less than or equal to 5. On exceeding this maximum value, the insertion leads to the following error:

```
INSERT INTO prods VALUES
(4, price_list (12,3,5,6,71,90));
ERROR at line 2:
ORA–22909: exceeded maximum VARRAY limit
```

Querying the data from varrays:

Data can be selected as a complete list from the table containing varrays. Individual manipulation cannot be done in SQL. Also, while performing updation, even if one single value has to be updated the whole collection has to be specified.

Example given below shows how to select and update records.

While issuing SELECT statement,

```
SELECT * FROM prods;
```

Displays,

```
PNO
-------
RATE
-------
2
RDBMS Concepts and Oracle 8i
358
PRICE_LIST(12, 34, 56)
1
PRICE_LIST(12, 34, 56, 78)
```

Selecting can be done only as a collection unit.

❑ **Check Your Progress – 4 :**

1. Each literal or column value manipulated by ________ has a datatype.

   a. Oracle b. SQL

2. Built– in ________ are predefined set of datatypes set in Oracle.

   a. Datatypes b. Database

3. ________ datatype is a fixed length character data of length n bytes.

   a. Char b. Cons

4. All elements of a given ________ are of the same datatype.

   a. Array b. Char

5. Oracle arrays are of variable size, which is why they are called ________.

   a. VARRAYs b. VRARRAYs

8.7 Declare Statements :

Variables can be declared in various ways. The two examples given above have shown how to declare variables and assign values to them. Certain variables can carry default values, which can be used to initialize values, and certain other variables can take constant values.

The following example illustrates the usage of the keywords DEFAULT and CONSTANT.

Using DEFAULT

```
DECLARE
S NUMBER DEFAULT 10;
```

```
BEGIN
DBMS_OUTPUT.PUT_LINE(s);
END
```

In this example, the default value of S i.e 10 is printed. The value in the variable s can be altered. It can also be re–assigned with any other numerical value.

Using CONSTANT

Consider a situation where variables have to contain constant values; like for example pi has to be assigned 3.14.

Example

```
DECLARE
Pi CONSTANT REAL:=3.14;
Area NUMBER;
R NUMBER:=&R;
BEGIN
Area:=pi*r**2;
DBMS_OUTPUT.PUT_LINE('THE AREA OF CIRCLE WITH RADIUS
'||r||' IS '||area);
END;
```

In the above example, the variable pi is declared as a constant variable whose value cannot change anywhere inside the program. Re–assigning the value for the same variable leads to an error.

Using NOT NULL

Besides assigning an initial value, declarations can impose the NOT NULL constraint, as the following example shows:

Example

```
DECLARE
s VARCHAR2(10) NOT NULL:='RADIANT';
BEGIN
DBMS_OUTPUT.PUT_LINE(s);
END;
```

The difference between using NOT NULL and DEFAULT is that, default can be assigned to NULL but NOT NULL as the name suggests cannot hold NULL values.

❑ **Check Your Progress – 5 :**

1. ________ can be declared in various ways.

   a. Variables   b. constants

2. ________ can be assigned to NULL but NOT NULL as the name suggests cannot hold NULL values.

   a. Default   b. Error

8.8 Lets Sum Up :

SQL is generally used for updating, deleting and querying information in Relational Database Management Systems (RDBMS). PL SQL is used for the shortcomings of SQL and enhances the characteristics of SQL. In this chapter, detailed discussion was made on PL/SQL. We also explain PL/SQL environment step by step, and also discussed variable, constant, datatypes & array with proper examples. And the structures associated with this procedural language extension to the Oracle database.

8.9 Answers for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a **2 :** b

❑ **Check Your Progress 2 :**

1 : a **2 :** a

❑ **Check Your Progress 3 :**

1 : a **2 :** a

❑ **Check Your Progress 4 :**

1 : a **2 :** a **3 :** a **4 :** a **5 :** a

❑ **Check Your Progress 5 :**

1 : a **2 :** a

8.10 Glossary :

Array : It is an ordered set of data elements.

8.11 Assignment :

Discuss use of Array in PL/SQL with suitable example

8.12 Activities :

Write a brief note on variables and constants.

8.13 Case Study :

Discuss the types of Datatypes.

8.14 Further Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw–Hill Publication.
3. Fundamentals of Database Systems by Ramez Elmsari, Shamkant B Navathe – Pearson Education
4. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
5. Database Management System by Raghu Ramkrishnan – Tata McGraw–Hill Publication.

Unit 09

CONTROL STATEMENT & INTERNAL FUNCTION

UNIT STRUCTURE

9.0 Learning Objectives :

After learning this block, you will be able to understand :

- Describe about Control Structures
- Describe Iteration Control
- Internal function used in SQL

9.1 Introduction :

Control statement specifies that other statements will be executed or not. PL/SQL support IF statement to control the execution of block of code. In decision–making scenarios, the condition statements like IF–THEN, IF–THEN–ELSE, IF–THEN–ELSEIF, and CASE are used.

In this chapter, we will discuss Loops used in PL/SQL. There is a situation when you need to execute a block of code multiple times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

A function can be called from inside a statement just like any other function (that is, by invoking the function's name), and can return a scalar value the built–in functions are recorded in a system table of the Postgre SQL database, pg_proc. As at Postgre SQL version 8.0 and later, this table have more than 1,700 entries. Inpsql, you can list all functions and their arguments using the \df command. Also, comments about any specific functions or group of functions can be displayed using the\dd command.

9.3 Control Statement :

Often it is necessary to take alternative actions depending on circumstances. The IF statement allows the execution of a sequence of statements conditionally. The execution purely depends on the value of the condition specified.

The working is similar to the IF conditions in other programming languages. There are three forms of IF statements :

IF…THEN

IF..THEN..ELSE

IF..THEN..ELSEIF

IF–THEN

This is the simplest form of the IF statement. These associates a condition with a sequence of statements enclosed by the keywords THEN and END IF. The following example illustrates this :

Syntax

```
IF condition THEN
Sequence_of_statements;
End if;
```

The sequence of statements is executed only if the condition yields TRUE. If the condition yields FALSE or NULL, the IF statement does nothing. In either case, control passes to the next statement following END IF. An example follows

Example

```
DECLARE
s number;
BEGIN
S:=&a;
If s>=10 THEN
DBMS_OUTPUT.PUT_LINE('S greater than or equal to 10');
END IF;
END;
```

The above example displays the value 'S greater than or equal to 10' only if the variables holds the value 10 or greater than that.

IF–THEN–ELSE

The second form of IF statement adds the keyword ELSE followed by an alternative sequence of statements. The general syntax is as follows.

```
IF condition THEN
Sequence_of_statements1;
ELSE

Sequence_of_statements2;

END IF;
```

The sequence of statements in the ELSE clause is executed only if the condition yields FALSE or NULL. Thus, the ELSE clause ensures that a sequence of statements is executed. Modifying the above example.

Example

```
DECLARE
S number; BegiN S:=&a;
IF S>=10 THEN
DBMS_OUTPUT.PUT_LINE('S GREATER THAN OR
EQUAL TO 10'); Else
DBMS_OUTPUT.PUT_LINE('S IS LESSER THAN 10');
END IF; END;
```

IF–THEN–ELSIF

The third form of IF statement uses the keyword ELSIF (not ELSIF) to introduce additional conditions. Multiple IF conditions are clubbed and written using the ELSIF clause.

Syntax

```
IF CONDITION1 THEN
SEQUENCE_OF_STATEMENTS1;
ELSIF CONDITION2 THEN
SEQUENCE_OF_STATEMENTS2;
ELSE
SEQUENCE_OF_STATEMENTS3;
END IF;
```

If the first condition yields FALSE or NULL, the ELSIF clause tests another condition. An IF statement can have any number of ELSIF clause; the final ELSE clause is optional. Conditions are evaluated one by one from top to bottom. If any condition yields TRUE, its associated sequence of statements is executed and control passes to the next statement following ENDIF.

If all the conditions yield FALSE or NULL, the sequence in the ELSE clause is executed. The following example finds the greatest of two numbers using IF–THEN–ELSIF.

Example

```
DECLARE
A NUMBER:=&A;
B NUMBER:=&B;
BEGIN
IF A>B THEN
DBMS_OUTPUT.PUT_LINE('THE GREATEST NUMBER IS '||A);
ELSIF B> A THEN
DBMS_OUTPUT.PUT_LINE('THE GREATEST NUMBER IS '||B);
ELSE
DBMS_OUTPUT.PUT_LINE('BOTH ARE EQUAL');
END IF;
END;
```

The above example accepts two numbers and the conditions are checked and depending on the condition that evaluates to TRUE, the message under that condition is displayed.

When possible, use the ELSIF clause instead of nested IF statements. This will make the code easier to read and understand. Compare the following IF statements.

```
IF condition1 THEN | IF condition1 THEN
statement1; | statement1;
ELSE | ELSIF condition2 THEN
IF condition2 THEN | statement2;
statement2; | ELSIF condition3 THEN
ELSE | statement3;
IF condition3 THEN | END IF;
statement3; |
END IF; |
END IF; |
END IF; |
```

These statements are logically equivalent. While the statement on the left obscures the flow of logic, the statement on the right reveals it.

Iterative Control :

Iterative statements are a set of statements that are performed a number of times depending on the value in the LOOP statement. There are three basic forms of LOOP statements

LOOP

- WHILE LOOP
- FOR LOOP

LOOP

The simplest form of LOOP statement is the basic (or infinite) loop, which encloses a sequence of statements between the keywords LOOP and END LOOP. The syntax is as follows.

Syntex

```
LOOP
Sequence_of_statements;
End LOOP.
```

With each Iteration of the loop, the sequence of statements is executed and the control resumes at the top of the loop. The following example shows the execution of the LOOP statement.

Example

```
DECLARE
a NUMBER:=10;
BEGIN LOOP
```

```
DBMS_OUTPUT.PUT_LINE(a);
END LOOP; END;
```

The output leads to an error. Because the number of times the loop must be executed is not specified and it goes for an infinite loop. In order to terminate the loop some form of termination statement is required. The EXIT statement is used to perform this operation.

Example

```
DECLARE
1. NUMBER:=&A;
BEGIN LOOP
DBMS_OUTPUT.PUT_LINE(A); a:=a+1;
IF a > 20
THEN
EXIT;
End if;
END LOOP
;
END;
```

Till the value crosses 20 the loop is executed and once the value for the variable reaches 20, the loop is terminated using the EXIT statement. This EXIT statement can be further simplified by the usage of EXIT_WHEN statement.

EXIT–WHEN

The EXIT–WHEN statement allows a loop to complete conditionally. When the EXIT when statement is encountered, the condition in the WHEN clause is evaluated. If the condition yields TRUE, the loop completes and control passes to the next statement after the loop. The syntax follows.

```
BEGIN
….
EXIT WHEN <CONDITION>;
END;
```

The following example shows the usage of EXIT WHEN statement. This example is a modification of the previous example with the EXIT statement.

Example

```
DECLARE
a NUMBER:=&a;
BEGIN
LOOP
DBMS_OUTPUT.PUT_LINE(a);
a:=a+1;
EXIT WHEN a>20;
```

```
END LOOP;
END;
```

While loop

The while loop statement associates a condition with a sequence of statements enclosed by the keywords LOOP and END LOOP. The syntax of the WHILE LOOP.

Syntax

```
While condition loop
Sequence_of_statements;
END LOOP;
```

Before each iteration of the loop, the condition is evaluated. If the condition yields TRUE, the sequence of statements is executed and the control resumes at the top of the loop. If the condition yields FALSE or NULL, the loop is bypassed and control passes to the next statement after end loop. The following example displays the total of the first 20 numbers using the WHILE LOOP statement.

Example

```
DECLARE
X integer:=1;
Y integer:=0;
BEGIN
WHILE x < = 20
LOOP Y:=Y+x; X:=X+1;
END LOOP;
DBMS_OUTPUT.PUT_LINE(Y);
END;
```

In above example, the loop is executed only when the while condition is satisfied.

FOR–LOOP

While the number of iterations for a WHILE loop is unknown until the loop completes, the number of iterations for a FOR loop is known before the loop is entered. FOR loops iterate over a specified range of integers. The range is part of an iteration scheme, which is enclosed within the keywords FOR and LOOPS. The syntax follows:

```
FOR counter IN [REVERSE] lower_bound..higher_bound LOOP
sequence_of_statements;
END LOOP;
```

The range is evaluated when the FOR loop is first entered and is never re–evaluated. As the next example shows, the sequence of statements is executed once for each integer in the range. After each iteration, the loop counter is incremented. The following example displays the reversal of a string:

Example

```
DECLARE
S VARCHAR2(20):
='&s'; S1
VARCHAR2(20);
BEGIN
FOR I IN 1..LENGTH(S)
LOOP
S1:=S1||SUBSTR(S,–I,1);
END LOOP;
DBMS_OUTPUT.PUT_LINE(s1);
END;
```

In the above example, a string is accepted. The number of iterations is fixed by the length of the string. Each character in the string is extracted and assigned to the variable S1. The variable is called a counter variable and it cannot be assigned or declared.

Example

```
DECLARE
s VARCHAR2(20):
='&s'; s1
VARCHAR2(20);
BEGIN
FOR i IN 1..LENGTH(s)

LOOP
s1:=s1||SUBSTR(s,– i,1);
DBMS_OUTPUT.PUT_LINE('The counter value is '||i);
END LOOP;

DBMS_OUTPUT.PUT_LINE(s1); END;
```

This displays the value of the counter variable.

By default iteration proceeds upward from the lower bound to the higher bound. However, the keyword REVERSE is used, iteration proceeds downward from the higher bound to the lower bound, as the example given below shows. After each iteration, the loop counter is decremented.

```
FOR i IN 1..3 LOOP -- assign the values 1,2,3 to i
sequence_of_statements; -- executes three times
END LOOP;
```

Never the less, the range bounds are to be written in ascending (not descending) order. Inside a FOR loop, the loop counter can be referenced like a constant. So, the loop counter can appear in expressions but it cannot be assigned values, as the following example shows:

```
FOR ctr IN 1..10 LOOP
...
IF NOT finished THEN
INSERT INTO ... VALUES (ctr, ...); -- legal
factor := ctr * 2; -- legal
ELSE
ctr := 10; -- illegal
END IF;
END LOOP;
```

Iteration schemes

The bounds of a loop range can be literals, variables or expressions; but they must evaluate to integers. For example, the following iteration schemes are legal:

```
j IN –5..5
k IN REVERSE first..last
step IN 0..TRUNC(high/low) * 2
code IN ASCII('A')..ASCII('J')
```

LOOP LABELS

Like PL/SQL blocks, loops can be labeled. The label, an undeclared identifier enclosed by double angle brackets, must appear at the beginning of the loop statement, as follows:

```
<<label_name>>
LOOP
Sequence_of_statements;
END LOOP;
```

Optionally, the label name can also appear at the end of the loop statement, as the following example shows:

```
<<my_loop>>
LOOP
….
END LOOP my_loop;
```

When labeled loops are nested, ending label names can be used to improve readability. With either form of EXIT statement, not only the current loop, but also any enclosing loops can be completed. This can be done by labeling the enclosing loop that is to be completed. The label can then be used in an EXIT statement, as follows

```
<<outer>>
LOOP
...
LOOP
...
```

```
Exit outer WHEN... exit both loops
End Loop;
...
End Loop outer;
```

Every enclosing loop up to and including the labeled loop is exited.

❑ **Check Your Progress – 1 :**

1. Control Statements is the ________ form of the IF statement.

   a. Simplest b. Complex

2. ________ statements are a set of statements that are performed a number of times depending on the value in the LOOP statement.

   a. Iterative b. Simple

9.4 Naming Convention :

The naming conventions apply to all PL/SQL program objects and units including constants, variables, cursors, exceptions, procedures, functions, and packages. Within the same scope, all declared identifiers must be unique. So, even if their datatypes differ, variables and parameters cannot share the same name.

In potentially ambiguous SQL statements, the names of local variables and formal parameters take precedence over the names of database tables. For example, the following SELECT statement fails because PL/SQL assumes that emp refers to the formal parameter:

```
PROCEDURE calc_bonus (emp NUMBER, bonus OUT REAL) IS
avg_sal REAL;
...
BEGIN
SELECT AVG(sal) INTO avg_sal FROM emp WHERE ...
...
END;
```

In such cases, you can prefix the table name with a username, as follows:

```
PROECEDURE calc_bonus (emp NUMBER, bonus OUT REAL) IS
avg_sal REAL;
...
BEGIN
SELECT AVG(sal) INTO avg_sal FROM scott.emp WHERE ...
...
END;
```

The names of database columns take precedence over the names of local variables and formal parameters. For example, the following DELETE statement removes all employees from the emp table, not just KING, because ORACLE assumes that both enames in the WHERE clause refer to the database column:

```
DECLARE
ename CHAR(10) := 'KING';
BEGIN
DELETE FROM emp WHERE ename = ename;
...
END;
```

In such cases, to avoid ambiguity, prefix the names of local variables and formal parameters with my_ as follows:

```
DECLARE
my_ename CHAR(10) := 'KING';
...
```

Or, use a block label to qualify references, as follows:

```
<<main>>
DECLARE
ename CHAR(10) := 'KING';
BEGIN
DELETE FROM emp WHERE ename = main.ename;
…
END;
```

The next example shows that you can use a supprogram name to qualify references to local variables and formal parameters:

```
PROCEDURE calc_bonus (empno NUMBER, bonus OUT REAL) IS
avg_sal REAL;
name CHAR(10);
job CHAR(15) := 'SALESMAN';
BEGIN
SELECT AVG(sal) INTO avg_sal FROM emp
WHERE job = calc_bonus.job; -- refers to local variable
SELECT ename INTO name FROM emp
WHERE empno = calc_bonus.empno; -- refers to parameter
…
END;
```

❑ Check Your Progress – 2 :

1. In potentially ambiguous SQL statements, the names of ________ and formal parameters take precedence over the names of database tables.

   a. local variables   b. Fixed

2. Within the same scope, all declared identifiers must be ________.

   a. Unique   b. Same

9.5 Internal Function :

Internal functions also known as Supprograms are named PL/SQL blocks that can take parameters and be invoked. PL/SQL has two types of supprograms called procedures and functions. Generally, a procedure is used to perform an action and a function to compute a value. Like unnamed or anonymous PL/SQL blocks, supprograms have a declarative part, an executable part, and an optional exception–handling part. The declarative part contains declarations of types, cursors, constants, variables, exceptions, and nested supprograms.

These items are local and cease to exist when you exit the supprogram. The executable part contains statements that assign values, control execution, and manipulate Oracle data. The exception–handling part contains exception handlers, which deal with exceptions raised during execution.

Advantages of Internal Functions / Supprograms :

Supprograms provide extensibility; that is, they let you tailor the PL/SQL language to suit your needs. Supprograms also provide modularity; that is, they let you break a program down into manageable, well–defined logic modules. This supports top–down design and the stepwise refinement approach to problem solving.

Also, supprograms promote reusability and maintainability. Once validated, a supprogram can be used with confidence in any number of applications. Furthermore, only the supprogram is affected if its definition changes. This simplifies maintenance and enhancement.

Finally, supprograms aid abstraction, the mental separation from particulars. To use supprograms, you must know what they do, not how they work. Therefore, you can design applications from the top down without worrying about implementation details. Dummy supprograms (stubs) allow you to defer the definition of procedures and functions until you test and debug the main program.

❑ **Check Your Progress – 3 :**

1. ________ provide extensibility; that is, they let you tailor the PL/SQL language to suit your needs.

   a. Supprograms b. Small programs

2. Supprograms provide________; that is, they let you tailor the PL/SQL language to suit your needs.

   a. Extensibility b. suitability

3. ________ promote reusability and maintainability.

   a. Supprograms b. Miniprograms

4. To use ________ you must know what they do, not how they work.

   a. supprograms, b. miniprograms

9.6 Let Us Sum Up :

In this chapter, detailed discussion was made on various decision control statements like IF…THEN, IF..THEN..ELSE and IF..THEN..ELSEIF. Discussion on various Looping statements used in PL/SQL like WHILE loop and FOR loop. We also learnt internal function also called subprogram. Many of the examples are given and discussed during discussions of various topics.

9.7 Answers for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a **2 :** a **3 :** a **4 :** a

❑ **Check Your Progress 2 :**

1 : a **2 :** a

❑ **Check Your Progress 3 :**

1 : a **2 :** a **3 :** a **4 :** a **5 :** a

9.8 Glossary :

Loop Statement : While implementing logic in PL/SQL, we may require to process a block of code repeatedly several times. This is achieved with the help of LOOP statements.

9.9 Assignment :

Discuss various IF statement used in PL/SQL.

9.10 Activities :

Write a note on Naming Convention.

9.11 Case Study :

Discuss internal function of PL/SQL

9.12 Furthers Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw–Hill Publication.
3. Fundamentals of Database Systems by Ramez Elmsari, Shamkant B Navathe –Pearson Education
4. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
5. Database Management System by Raghu Ramkrishnan – Tata McGraw–Hill Publication.
6. SQL, PL/SQL : The Programming Language Oracle – Ivan Bayross – BPB Publication.

Unit 10

STORED PROCEDURE

UNIT STRUCTURE

10.0 Learning Objectives :

After learning this block, you will be able to understand:

- The types of stored procedures
- The Create Procedure command
- Deleting or replacing a stored procedure
- Use of Commit Statement

10.1 Introduction :

A stored procedure is a combination of SQL statements and control and condition handling statements that provides an interface to the Teradata Database.The term stored procedure refers to a stored procedure you write with SQL statements. The term external stored procedure refers to a stored procedure you write in C, C++, or Java.A stored procedure is a database object executed on the Teradata Database. Typically, a stored procedure consists of a procedure name, input and output parameters, and a procedure body. See the next subsection for information on the procedure body. Use the CREATE PROCEDURE statement to define a stored procedure.

For details, see —CREATE/REPLACE PROCEDURE|| in SQL Data Definition Language. For each stored procedure, the database includes a stored procedure table that contains the stored procedure body you write and the corresponding compiled stored procedure objectcode. Data dictionary tables contain stored procedure parameters and attributes.

10.3 Stored Procedure :

The invocation of a stored procedure is treated as a regular external call. The application waits for the stored procedure to terminate, and parameters can be passed back and forth. Stored procedures can be called locally (on the same system where the application runs) and remotely on a different system. However, stored procedures are particularly useful in a distributed environment since they may considerably improve the performance of distributed applications by reducing the traffic of information across the communication network.

For example, if a client application needs to perform several database operations on a remote server, you can choose between issuing many different database requests from the client siteand calling a stored procedure. In the first case, you start a window with the remote system every time you issue a request.

If you call a stored procedure instead, only the call request and the parameters flow on the line. In addition, the server system executes some of the logic of your application with potential performance benefits at the client site. Your programming productivity can be improved by using stored procedures when you develop distributed applications. Stored procedures are the easiest way to perform a remote call and to distribute the execution logic of application program. Stored procedures can be used for many different application purposes such as :

- Distributing the logic between a client and a server
- Performing a sequence of operations at a remote site
- Combining results of query functions at a remote site
- Controlling access to database objects
- Performing non–database functions

Let us look at a typical example where stored procedures can be effective. A company runsits business on a server located at the headquarters and on client systems located at every branch office. A user at a branch office is working with an invoice clearance application, which has to update three tables on the server :

- The invoice table is named INVOICE.
- The customer table is named CUSTOMER.
- The account receivable balance table is named ARBLNCE.

An invoice record is flagged with a ―clearance‖ marker, and after that the corresponding CUSTOMER record is updated by deducting the invoice amount from the current account receivable total amount. Finally, the account receivable balance record must be updated as well. Figure 10.1 shows a distributed application for the invoice clearance process that was implemented without resorting to stored procedures. The client system has to access the server database several times for every update event, sending and receiving data across communication lines for every request. In addition, all the application logic is implemented at the client site.

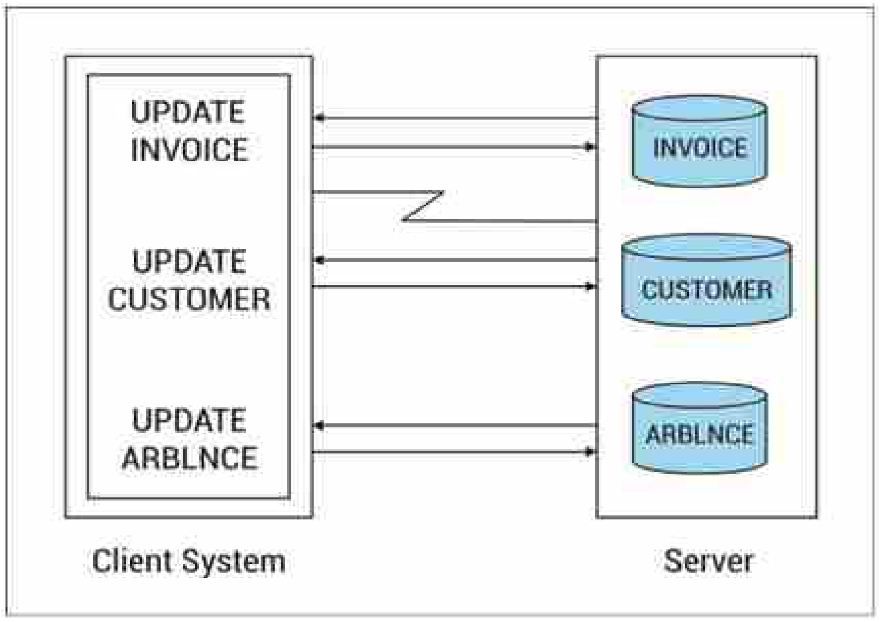


Fig. 10.1 Distributed Applications Without Stored Procedures

Figure 10.2 shows how we can take advantage of stored procedures in developing this application.

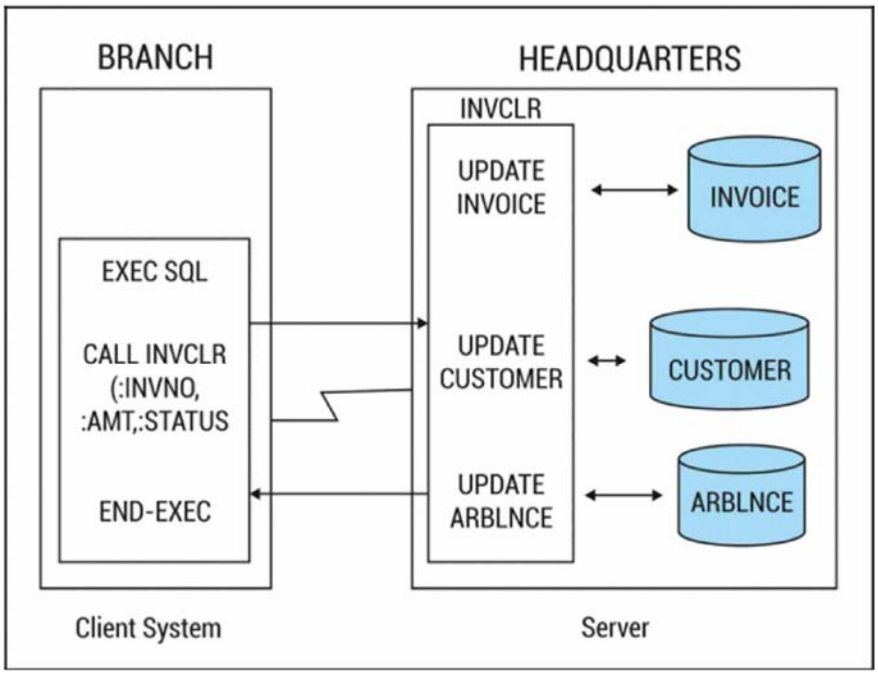


Fig. 10.2 Distributed Application With Stored Procedures

The same application functions can be carried out by calling a single stored procedure that runs at the server site. The communications window is greatly reduced, and the network resources are better balanced by splitting the application logic.

Modularity in application development is also encouraged by using stored procedures. This makes application maintenance easier and improves code reusability. It is useful to compare stored procedures to other tools and techniques for distributed application development such as DRDA SQL, DDM Submit Remote Command (SBMRMTCMD), and triggers :

- With DRDA SQL, the application logic is fully implemented at the application requestersite. Stored procedures are the natural extension for DRDA applications, since they allow you to easily split the application logic.
- The Submit Remote Command (SBMRMTCMD) command submits a CL command using Distributed Data Management (DDM) support to run on the target system. The SBMRMTCMD command allows a user at a client system to perform some object management operations rather than running remote applications. You may also want to submit user–written commands or programs to run on the target system, but you face the following restrictions:

The target (server) system cannot send any parameters to the source (client) system. Only a generic return code is sent back to signal whether the remote execution completed successfully.

Any changes in database tables made by the server application on the server system cannot be committed or rolled back by the client application.

- Triggers are user–written programs associated with a table. Unlike stored procedures, they are almost independent from applications because they are automatically executed either before or after a database change. Stored procedures need to be called explicitly by the SQL CALL statement.

Triggers receive from the database manager a standard parameter list, which is input only, and they cannot pass any information back to the application through the parameter list. Therefore, when the trigger ends abnormally, the application must receive an error message or an SQLCODE and handle it. Stored procedures can receive input/output parameters and use them to communicate with the client application.

Triggers can be used to enforce business rules. Stored procedures are used mainly to improve the performance of distributed applications and productivity of application development. Stored procedures are capable of returning result sets, which makes them very flexible and efficient in client/ server environments.

10.3.1 Stored Procedure Types :

There are two categories into which stored procedures can be divided:

- SQL stored procedures
- External stored procedures

10.3.2 SQL Stored Procedures :

SQL stored procedures are written in the SQL language. This makes it easier to port stored procedures from other database management systems (DBMS) to the iSeries server andfrom the iSeries server to other DBMS. Implementation of the SQL stored procedures isbased on procedural SQL standardized in SQL99.

10.3.3 External Stored Procedure :

An external stored procedure is written by the user in one of the programming languages on the iSeries server. You can compile the host language programs to create *PGM objects or Service Program. To create an external stored procedure, the source code for the host language must be compiled so that a program object is created. Then the CREATE PROCEDURE statement is used to tell the system where to find the program object that implements this stored procedure. The stored procedure registered in the following example returns the name of the supplier with the highest sales in a given month and year. The procedure is implemented in ILE RPG with embedded SQL :

```
c+ CREATE PROCEDURE HSALE
c+ (IN YEAR INTEGER ,
c+ IN MONTH INTEGER ,
c+ OUT SUPPLIER_NAME CHAR(20) ,
c+ OUT HSALE DECIMAL(11,2))
c+ EXTERNAL NAME SPROCLIB.HSALES
c+ LANGUAGE RPGLE
c+ PARAMETER STYLE GENERAL
c/END_EXEC
```

The following SQL CALL statement calls the external stored procedure, which returns a supplier name with the highest sales :

```
c/EXEC SQL
c+ CALL HSALE(:PARM1, :PARM2, :PARM3, :PARM4)
c/END–EXEC
```

An external stored procedure may contain no SQL statements. For example, you may createa stored procedure that uses the native interface to access the DB2 Universal Database for iSeries data.

10.3.4 Java Stored Procedures :

Java stored procedures were first introduced in the iSeries server starting with V4R5, as a particular case of external stored procedures limited to not being able to return result sets. Starting with V5R1, the support of result sets was added to Java stored procedures. There is a growing interest in Java and its portability across platform that makes Java stored procedures a very interesting option to consider.

10.3.5 Registering Stored Procedures :

Before a stored procedure can be called by a client program, it must be registered with the database using the DECLARE PROCEDURE or the CREATE PROCEDURE statement. The stored procedure can also be defined using either of these statements. The CREATE PROCEDURE statement differs from the DECLARE PROCEDURE since it adds procedure and parameter definitions to the system catalog tables (SYSROUTINES and SYSPARMS).This way, a stored procedure becomes available for any client program running on the local orthe remote system. Since the information about the stored procedure is stored in the systemcatalog tables, the CREATE PROCEDURE needs to be performed only once in the lifetime of a stored procedure. Use the DROP PROCEDURE statement to delete the stored procedure Catalog information entry. The DECLARE PROCEDURE statement is not frequently used. It is mainly for temporary registration of stored procedures.

➢ **Create Procedure :**

The CREATE PROCEDURE statement can be used to create any of the two types of stored procedures. This statement can be issued interactively, or it can be embedded in an application program. After a procedure is registered, it can be called from any interface supporting the SQL CALL statement.

During stored procedure creation, you can control characteristics that affect the way the stored procedure is identified in DB2 Universal Database for iSeries or its behavior. This section explains some of them.

➢ **SPECIFIC Specific–Name :**

DB2 Universal Database for iSeries identifies each stored procedure with a specific name that, combined with the specific schema, must be unique in the system. This gains importance because multiple stored procedures with the same name but different signatures must have different specific names.If you do not provide a specific name, DB2 Universal Database for iSeries generates one automatically. If the SQL procedure name is longer than 10 characters, this name can be used to specify the C program name instead of having DB2 enerate one automatically, as shown in the following example:

```
CREATE PROCEDURE SAMPLE.ALLOCATECOSTS(...)
...
SPECIFIC ALLOCATECOSTS_3PARMS
```

❑ **Check Your Progress – 1 :**

1 The invocation of a ________ is treated as a regular external call.

a. stored procedure b. create procedure

2. The client system has to access the server database ________ for every update event.

a. several times b. once

3. ________ can be used to enforce business rules.

a. Triggers b. Guns

4. ________ procedures are used mainly to improve the performance of distributed applications and productivity of application development.

a. Stored b. saved

5. An external stored procedure is written by the user in one of the programming languages on the ________ server.

a. iSeries b. SQL

10.4 Overloading :

Postgre SQL considers functions to be distinct if they have different names, if they have adifferent number of parameters, or if their parameters have different types. We can create further add_one functions that deal with different types if we wish. Consider what happens when we use our add_one function on a floating–point value :

```
bpfinal=# SELECT add_one(3.1);
```

ERROR : function add_one(numeric) does not exist

HINT : No function matches the given name and argument types. You may need to add explicit type casts.

```
bpfinal=#
```

Postgre SQL returns an error, because it could not locate a version of the add_one function that takes a floating–point value as its parameter.

Note : Earlier versions of Postgre SQL would have executed this add_one function by automatically converting the floating–point value to an integer. The value 3.1 would have been rounded down to 3, and the function would have returned 4.

If we wish to have an increment function for floating–point numbers, we just need to create another definition of add_one :

```
bpfinal=# CREATE FUNCTION
bpfinal–# add_one(float8)RETURNS float8
bpfinal–# AS =
bpfinal'# BEGIN
bpfinal'# RETURN $1 + 1;
bpfinal'# END;
bpfinal'# =
bpfinal–# LANGUAGE =plpgsql';
CREATE FUNCTION
bpfinal=# SELECT add_one(3.1);
add_one
--------
4.1
(1 row)
bpfinal=#
```

This time, we get the result we want because Postgre SQL can find and execute an appropriate version of the add_one function. This behavior, known as function over loading, can be quite useful, but also fairly confusing. To keep the functions distinct, we must refer to them in a way that indicates their parameters. In this case, we have two functions that we can refer toas add_one(int4) and add_one(float8).

❑ Check Your Progress – 2 :

1. ________ considers functions to be distinct if they have different names.

   a. PostgreSQL b. MSQL

2. We can create further ________ functions that deal with different types if we wish.

   a. add_one b. many

10.5 Commit Statement :

As a SQL language we use transaction control language very frequently. Committing a transaction means making permanent the changes performed by the SQL statements with in the transaction. A transaction is a sequence of SQL statements that Oracle Database treats as a single unit. This statement also erases all save points in the transaction and releases transaction locks.

Oracle Database issues an implicit COMMIT before and after any data definition language (DDL) statement. Oracle recommends that you explicitly end every transaction in your application programs with a COMMIT or ROLLBACK statement, including the last transaction, before disconnecting from Oracle Database. If you do not explicitly commit the transaction and the program terminates abnormally, then the last uncommitted transaction is automatically rolled back.

➢ **Until you Commit a Transaction :**

- You can see any changes you have made during the transaction by querying the modified tables, but other users cannot see the changes. After you commit the transaction, the changes are visible to other users' statements that execute after the commit
- You can roll back (undo) any changes made during the transaction with the ROLLBACK statement

Note : Most of the people think that when we type commit data or changes of what you have made has been written to data files, but this is wrong when you type commit it means that you are saying that your job has been completed and respective verification will be done by oracle engine that means it checks whether your transaction achieved consistency when it finds ok it sends a commit message to the user from log buffer but not from data buffer, so after writing data in log buffer it insists data buffer to write data in to data files, this is how it works.

Before a transaction that modifies data is committed, the following has occurred :

- Oracle has generated undo information. The undo information contains the old data values changed by the SQL statements of the transaction
- Oracle has generated redo log entries in the redo log buffer of the System Global Area (SGA). The redo log record contains the change to the datablock and the change to the rollback block. These changes may go to disk before a transaction is committed
- The changes have been made to the database buffers of the SGA. These changes may go to disk before a transaction is committed.

Note : The data changes for a committed transaction, stored in the database buffers of the SGA, are not necessarily written immediately to the data files by the database writer (DBWn) background process. This writing takes place when it is most efficient for the database to do so. It can happen before the transaction commits or, alternatively, it can happen some times after the transaction commits.

When a transaction is committed, the following occurs :

1. The internal transaction table for the associated undo table space records that the transaction has committed, and the corresponding unique system change number (SCN) of the transaction is assigned and recorded in the table
2. The log writer process (LGWR) writes redo log entries in the SGA's redo log buffers to the redo log file. It also writes the transaction's SCN to the redo log file. This atomic event constitutes the commit of the transaction
3. Oracle releases locks held on rows and tables
4. Oracle marks the transaction complete

The syntax of Commit Statement is

COMMIT [WORK] [COMMENT byour comment'];

WORK is optional.

The WORK keyword is supported for compliance with standard SQL. The statements COMMIT and COMMIT WORK are equivalent.

Examples

Committing an Insert

INSERT INTO table_name VALUES (val1, val2);

COMMIT WORK;

COMMENT

Comment is also optional. This clause is supported for backward compatibility. Oracle recommends that you used named transactions instead of commit comments. Specify a comment to be associated with the current transaction. The ‗text' is a quoted literal of up to 255 bytes that Oracle Database stores in the data dictionary view DBA_2PC_PENDING along with the transaction ID if a distributed transaction becomes in doubt. This comment can help you diagnose the failure of a distributed transaction.

Examples

The following statement commits the current transaction and associates a comment with it :

COMMIT

COMMENT ‗In–doubt transaction Code 36, Call (415) 555–2637';

➢ **WRITE Clause :**

Use this clause to specify the priority with which the redo information generated by the commit operation is written to the redo log. This clause can improve performance by reducing latency, thus eliminating the wait for an I/O to the redo log. Use this clause to improve response time in environments with stringent response time requirements where the following conditions apply :

The volume of update transactions is large, requiring that the redo log be written to disk frequently. The application can tolerate the loss of an a synchronously committed transaction.

The latency contributed by waiting for the redo log write to occur contributes significantly to overall response time.

You can specify the WAIT | NOWAIT and IMMEDIATE | BATCH clauses in any order.

Examples

To commit the same insert operation and instruct the database to buffer the change to the redo log, without initiating disk I/O, use the following COMMIT statement :

COMMIT WRITE BATCH;

Note : If you omit this clause, then the behavior of the commit operation is controlled by the COMMIT_WRITE initialization parameter, if it has been set. The default value of the parameter is the same as the default for this clause. Therefore, if the parameter has not been set and you omit this clause, then commit records are written to disk before control is returned to the user.

WAIT | NOWAIT Use these clauses to specify when control returns to the user.

The WAIT parameter ensures that the commit will return only after the corresponding redo is persistent in the online redo log. Whether in BATCH or IMMEDIATE mode, when the client receives a successful return from this COMMIT statement, the transaction has been committed to durable media. A crash occurring after a successful write to the log can prevent the success message from returning to the client. In this case the client cannot tell whether or not the transaction committed.

The NOWAIT parameter causes the commit to return to the client whether or not the write to the redo log has completed. This behavior can increase transaction through put. With the WAIT parameter, if the commit message is received, then you can be sure that no data has been lost.

Caution :

With NOWAIT, a crash occurring after the commit message is received, but before the redo log record(s) are written can falsely indicate to a transaction that its changes are persistent. If you omit this clause, then the transaction commits with the WAIT behavior.

IMMEDIATE | BATCH Use these clauses to specify when the redo is written to the log.

The IMMEDIATE parameter causes the log writer process (LGWR) to write the transaction's redo information to the log. This operation option forces a disk I/O, so it can reduce transaction through put.

The BATCH parameter causes the redo to be buffered to the redo log, along with other concurrently executing transactions. When sufficient redo information is collected, a disk write of the redo log is initiated. This behavior is called —group commit||, as redo for multiple transactions is written to the log in a single I/O operation.

If you omit this clause, then the transaction commits with the IMMEDIATE behavior.

FORCE Clause

Use this clause to manually commit an in–doubt distributed transaction or a corrupt transaction.

In a distributed database system, the FORCE string [,integer] clause lets you manually commit an in–doubt distributed transaction. The transaction is identified by the ‗string' containing its local or global transaction ID. To find the IDs of such transactions, query the data dictionary view DBA_2PC_PENDING. You can use integer to specifically assign the transaction a system change number (SCN). If you omit integer, then the transaction is committed using the current SCN.

The FORCE CORRUPT_XID ‗string' clause lets you manually commit a single corrupt transaction, where string is the ID of the corrupt transaction. Query the V$CORRUPT_XID_LIST data dictionary view to find the transaction IDs of corrupt transactions. You must have DBA privileges to view the V$CORRUPT_XID_LIST and to specify this clause.

Specify FORCE CORRUPT_XID_ALL to manually commit all corrupt transactions. You must have DBA privileges to specify this clause.

Examples

Forcing an in doubt transaction. Example The following statement manually commits a hypothetical in–doubt distributed transaction. Query the V$CORRUPT_XID_LIST data dictionary view to find the transaction IDs of corrupt transactions. You must have DBA privileges to view the V$CORRUPT_XID_LIST and to issue this statement.

COMMIT FORCE ‗22.57.53';

❑ **Check Your Progress – 3 :**

1. As a SQL language we use ________ control language very frequently.

   a. Transaction b. Function

2. A ________ is a sequence of SQL statements that Oracle Database treats as a single unit.

   a. Transaction b. function

3. The ________ parameter causes the commit to return to the client whether or not the write to the redo log has completed.

   a. NOWAIT b. IMMEDIATE

4. The ________ parameter causes the log writer process (LGWR) to write the transaction's redo information to the log.

   a. IMMEDIATE b. NOWAIT

5. The ________ parameter causes the redo to be buffered to the redo log, along with other concurrently executing transactions.

   a. BATCH b. NOWAIT

10.6 Let Us Sum Up :

In this unit, we have learnt so much about the stored procedure and functions; we learnt that stored procedure and functions are sub programs stored in database. Functions return a single value whereas procedure doesn't return any value. Stored procedures have important advantages such as improving performance, making maintenance and security implementation easy.

We learnt that Oracle allows parameters of three types – in, out, and in out.

OUT and IN OUT parameters are used to return values back to calling program.

They can be passed by either value (default) or by reference using NOCOPY hint.

We even learnt that Procedures are executed under privileges of owner of the procedure. However, it is possible to execute procedure with the privileges of invoker using AUTHID CURRENT_USER option of CREATE PROCEDURE command. Standard procedure RAISE_APPLICATION_ERROR is used to raise an application error with the given error number and message.

10.7 Answers for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a **2 :** a **3 :** a **4 :** a **5 :** a

❑ **Check Your Progress 2 :**

1 : a **2 :** a

❑ **Check Your Progress 3 :**

1 : a **2 :** a **3 :** a **4 :** a **5 :** a

10.8 Glossary :

Committing a Transaction : It means making permanent the changes performed y the SQL statements within the transaction.

10.9 Assignment :

1. Discuss the stored procedures and also tell about their purposes.
2. Discuss the types of stored procedures.

10.10 Activities :

1. Discuss the procedure of creating procedures.
2. Discuss registering stored procedures.

10.11 Case Study :

Explain commit statement.

10.12 Furthers Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw–Hill Publication.
3. Fundamentals of Database Systems by Ramez Elmsari, Shamkant B Navathe –Pearson Education
4. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
5. Database Management System by Raghu Ramkrishnan – Tata McGraw–Hill Publication.
6. SQL, PL/SQL : The Programming Language Oracle – Ivan Bayross – BPB Publication.

Unit 11

DATABSE TRIGGER & PACKAGES

UNIT STRUCTURE

11.0 Learning Objectives :

After learning this unit, you will be able to understand :

- Database triggers and their use
- Create database triggers
- Database trigger firing rules
- How to Remove database triggers
- How to create Package

11.1 Introduction :

In our stored procedure example, we developed a function that would allow us to discover which products required restocking. The function wrote reminder messages into a reorders table. For this to be useful, we would need to ensure that this procedure is executed on a regular basis, perhaps once a day during an overnight batch process. It might be an advantage to find a way of automatically making sure the reorders table is always up–to–date, without needing to program our client applications to keep updating the entries.

In our earlier discussion, we mentioned the concept of referential integrity– ensuring that the data in our database makes sense at all times. For example, if we delete a customer, we need to ensure that all of the order history relating to that customer is deleted at the same time. We have seen constraints used to ensure that Postgre SQL enforces this kind of integrity. For some applications, constraints are not quite enough. Suppose we want to prevent the deletion of a customer when there are still outstanding orders for that customer, but allow the deletion if all orders have been shipped.

Earlier we saw how column–level and table–level constraints could be used to enforce more complex rules of data integrity, but these rules were essentially static. We could specify that a related row must exist, or enforce a rule that you cannot delete while related rows exist.

However, we had no way of specifying complex conditions, such as that a row must not exist unless some other condition is also true. Nor could we carry out more complex user–defined actions when rows were added or deleted.

One solution to these problems is the use of triggers. With a trigger, we can arrange for Postgre SQL to execute a stored procedure when certain actions are taken, like an INSERT, DELETE, or UPDATE in a table.

The combination of stored procedures and triggers gives us the power to enforce quite sophisticated business rules directly in the database. As discussed earlier, that the best place for enforcing business rules about the data is in the database. To use a trigger, we need to first define a trigger procedure. Then we create the trigger itself, which defines when the trigger procedure will be executed.

11.3 Database Trigger :

A database trigger is a stored PL/SQL block that is associated with a table. Triggers are automatically executed when a specified SQL statement is issued against the table. Triggers are mainly used for the following purposes :

- To automatically generate values
- To provide auditing.
- To prevent invalid transactions.
- To check for complex integrity constraints that cannot be given in Constraints.
- To maintain replicate tables.

Triggers contain PL/SQL constructs and they can be fired only for DML Statements like INSERT, UPDATE and DELETE. Triggers are event based.

Parts of a Trigger

A Trigger contains 3 parts. They are

- Triggering event or statement
- Trigger Restriction
- Trigger Action

Triggering Event or Statement

It is a SQL statement that causes the trigger to be fired. The valid statements are the DML statements – Insert, Update or Delete. Additionally, trigger can be made to be fired when a particular column is updated. Also it can contain multiple DML statements.

Triggering Restriction

This is a Boolean expression which must be TRUE for the trigger to be fired.

Trigger Action

This is a procedure containing SQL and PL/SQL statements to be executed when a triggering statement is issued and also the triggering restriction is true.

Types of Triggers

Triggers are classified into different types depending on when the trigger is to be fired.

They are :

- BEFORE
- AFTER
- INSTEAD OF (dealt later in this chapter)

BEFORE / AFTER

The BEFORE option of a trigger is used to specify when the trigger must be fired. If the option BEFORE is chosen, the trigger is fired before the triggering statement is executed. In the case of AFTER, the trigger is fired after executing the statement.

When to Use Before and After

BEFORE

Before triggers are used when the triggering action determines whether the triggering statement must be allowed to be completed or not. This eliminates un–necessary processing of the triggering statement. Before triggers can also be used to derive specific column values before completing an Insert or Update statement ?

AFTER

This trigger is executed after the trigger statement is issued. It is used when the trigger statement has to be completed before the trigger action. If a Before Trigger is already present, an after trigger can be used to perform different actions on the same statement. Based on how many records are to be affected by the triggers, triggers can be classified as;

- Row Level
- Statement Level

Row Level Triggers

These types of triggers are fired for each row that is affected by the triggering statement. For example, UPDATE statement for multiple rows of a table. In this case, the ROW level triggers are executed for every row that is affected by the UPDATE statement.

Statement Level Triggers

This trigger is fired once on behalf of the triggering statement regardless of the number of rows in the table that the triggering statement affects. Based on the combinations of the above, there are twelve types of triggers that can be written on a table.

Statement Level	**Row Level**
Before Insert	Before Insert
Before Update	Before Update
Before Delete	Before Delete
After Insert	After Insert
After Update	After Update
After Delete	After Delete

❑ **Check Your Progress – 1 :**

1. A ________ is a stored PL/SQL block that is associated with a table.

   a. Database Trigger b. Stored Programs

2. It is a ________ that causes the trigger to be fired.

   a. SQL Statement b. Database

3. The ________ option of a trigger is used to specify when the trigger must be fired.

   a. Before b. After

4. ________ trigger is executed after the trigger statement is issued.

   a. After b. before

5. ________ is fired once on behalf of the triggering statement regardless of the number of rows in the table that the triggering statement affects.

   a. Statement Level b. Program Level

11.4 Creation of Database Trigger :

Triggers are created using CREATE TRIGGER statement. The name of the trigger must be unique with respect to other triggers in the same schema. The syntax for creating the trigger is :

Syntax:

```
CREATE OR REPLACE TRIGGER <triggername> [BEFORE/AFTER]
[INSERT/UPDATE/DELETE] ON <tablename> [FOR EACH ROW]
[WHEN <condition>]
```

The following example creates a simple trigger.

Example

```
CREATE OR REPLACE TRIGGER instremp BEFORE INSERT ON emp
BEGIN
DBMS_OUTPUT.PUT_LINE(?Inserting Record');
END;
/
```

The trigger gets created.

When an Insert statement is issued, the output will look like :

```
INSERT INTO emp (empno,ename,deptno) VALUES (1,'Giri',10);
```

This would display

```
Inserting Record
1 row created.
```

In the above example, the triggering statement is

BEFORE INSERT ON emp

Hence the message ‗Inseting record is displayed before the row is created

There is no trigger restriction the body or action of the trigger starts from the Begin statement.

Example

This trigger displays the total number of records available in the table if insertion is done.

CREATE OR REPLACE TRIGGER INSTR1 BEFORE INSERT ON EMP DECLARE

T NUMBER; BEGIN

SELECT COUNT(*) INTO X FROM EMP;

DBMS_OUTPUT.PUT_LINE(X+1 ||' NUMBER OF RECORDS ARE AVAILABLE');

END;

While insertion is done, the output will look like :

insert into emp(empno,ename,deptno) values(12,'hari',30); 15 number of records are available

Before insertion the total records were 14. After insertion the total records increases to 15.

Note here, there is an explicit DECLARE statement and no IS/AS is specfiied. If a trigger containsany variable, DECLARE must be specified.

The trigger execution is performed in the following ways :

- All the Before statement triggers are executed.
- The Before Row Level triggers are executed.
- After Row Level triggers are executed.
- After Statement triggers are executed that apply to the statement.

Accessing Column Values

When a trigger action or the trigger body contains statements that require access to the table values or for checking the new value with the old value, two correlation names are used :

:new and :old. :new refers to the new values entered in the trigger statement and :old refers to the old existing value in the table. These are also called as pseudo records since they do not contain any permanent record. Also these can be given only for row–level triggers. The following table illustrates the values held by these with respect to the DML operations.

Statement	**:new**	**:old**
Insert	New values entered in the	Null
Update	Insert command New values entered in Update statement	Old values available in the table
Delete	Null	Old values in the table

They are available for both before and after triggers.

Example

This example uses the: new and displays the newly inserted record.

CREATE OR REPLACE TRIGGER INSTR2 AFTER INSERT ON DEPT FOR EACH ROW

```
BEGIN
DBMS_OUTPUT.PUT_LINE(:NEW.DEPTNO||' =||:NEW.DNAME||'
=||:NEW.LOC);
END;
```

The output will look like :

```
INSERT INTO DEPT VALUES(12,'SALES','CHENNAI');
12 SALES CHENNAI
1 row created.
```

The following example checks for the dept no availability in the dept table. If the corresponding dept no is not available in the table it displays a message as shown in the following example.

```
CREATE OR REPLACE TRIGGER CHKDEPTNO BEFORE UPDATE
OF DEPTNO ON EMP FOR EACH
ROW
DECLARE
OLD_DNO NUMBER;
BEGIN
SELECT DEPTNO INTO OLD_DNO FROM DEPT WHERE
DEPTNO=:NEW.DEPTNO;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE(?DEPTNO NOT AVAILABLE');
END;
```

The output of the program

```
UPDATE EMP SET DEPTNO=34 WHERE EMPNO=7900;
DEPTNO NOT AVAILABLE
1 row updated.
```

Using Referencing Option

The Referencing option can be specified in a trigger body of a row trigger to avoid name conflicts among the correlation names and tables that might be named.

Using Conditional Predicates

If more than one type of DML operation can fire a trigger, all the operations can be placed inside a single trigger each differentiated by what are called conditional predicates.

For referring to Insert operation, INSERTING is used, for Delete operation, DELETING and for update operation, UPDATING predicates are used. These are Boolean values which return TRUE only when the statement that fired a trigger is any one of DML statements. The syntax is given below :

```
IF INSERTING THEN…..
ELSIF UPDATING THEN …..
ELSIF DELETING THEN .....
      END IF;
```

Example

```
CREATE OR REPLACE TRIGGER prdtrig AFTER INSERT OR UPDATE
OR DELETE ON
product
BEGIN
IF INSERTING THEN
DBMS_OUTPUT.PUT_LINE(‗Inserting operation');
ELSIF UPDATING THEN
DBMS_OUTPUT.PUT_LINE(‗Updating operation');
ELSIF DELETING THEN
DBMS_OUTPUT.PUT_LINE(‗Deleting operation');
END IF;
END;
```

Using Raise_Application_Error

As we have already seen in exceptions, Raise_Application_Error terminates the operation abruptly and comes to the SQL prompt. Now, consider a situation such that, a record gets inserted and the trigger based on that causes another insert on another table.

For example consider the following trigger. This trigger checks for the job and the corresponding salary in the table. If there is any violation, it reports to the user.

```
CREATE OR REPLACE TRIGGER CHKSALJOB AFTER INSERT ON
EMP FOR EACH ROW
BEGIN
IF NOT(:NEW.JOB=?MANAGER' AND (:NEW.SAL > 4000 AND
:NEW.SAL <6000))
THEN
DBMS_OUTPUT.PUT_LINE(?Limit for Manager is between 4000 and
6000');
END IF;
END;
```

The trigger gets created. When a record is inserted on to the table EMP which violates the condition the message is displayed as

```
INSERT INTO emp (empno,ename,deptno,mgr,job,sal) VALUES
(1234,'ARUN',20,7902,'MANAGER',8000); Limit for Manager is between
4000 and 6000
1 row created.
```

Even though the message is displayed, still this does not stop the record from being inserted.

Now let us try to raise this as an exception with RAISE statement and the code is re–written as follows :

```
CREATE OR REPLACE TRIGGER CHKSAL JOB AFTER INSERT ON EMP FOR EACH ROW
DECLARE
raise_ex EXCEPTION;
BEGIN
IF NOT(:NEW.JOB=‗MANAGER' AND (:NEW.SAL > 4000 AND :NEW.SAL <6000))
THEN
RAISE raise_ex;
END IF;
EXCEPTION
WHEN RAISE_EX THEN
DBMS_OUTPUT.PUT_LINE(‗Limit for Manager is between 4000 and 6000');
END;
```

The trigger is created.

After this, when the insert statement is issued,

```
INSERT INTO emp(empno,ename,deptno,mgr,job,sal) VALUES
(1234,'ARUN',20,7902,'MANAGER',8000);
Limit for Manager is between 4000 and 6000
1 row created.
```

Even this does not stop from inserting the record onto the table.

RAISE_APPLICATION_ERROR is used to perform this operation. The following example does not insert record into the table if the condition is not satisfied.

Example

```
CREATE OR REPLACE TRIGGER CHKSAL JOB AFTER INSERT ON EMP FOR EACH ROW
BEGIN
IF NOT(:NEW.JOB=?MANAGER' AND (:NEW.SAL > 4000 AND :NEW.SAL <6000))
THEN
RAISE_APPLICATION_ERROR(–20000,'MANAGER SALARY CANNOT EXCEED 8000 AND CANNOT BE LESS THAN 4000');
END IF;
END;
```

The output when record is inserted is :

```
INSERT INTO emp(empno,ename,deptno,mgr,job,sal) VALUES
(1234,'ARUN',20,7902,'MANAGER',8000);
```

INSERT INTO emp(empno,ename,deptno,mgr,job,sal) VALUES (1234,'ARUN',20,7902,'MANAGER',8000)

*

ERROR at line 1:

ORA–20000: Manager Salary cannot exceed 8000 and cannot be less than

4000

ORA–06512: at ―HEMA.CHKSALJOB‖, line 4

ORA–04088: error during execution of trigger ‗HEMA.CHKSALJOB'

❑ **Check Your Progress – 2 :**

1. Triggers are created using ________ statement.

   a. CREATE TRIGGER b. START TRIGGER

2. ________ displays the total number of records available in the table if insertion is done.

   a. Trigger b. Funciton

3. The ________ option can be specified in a trigger body of a row trigger to avoid name conflicts among the correlation names and tables that might be named.

   a. Referencing b. Trigger

4. ________ terminates the operation a?ruptly and comes to the SQL prompt.

   a. Raise_Application_Error b. Application_Error

11.5 Implementing Trigger :

Enabling and Disabling Triggers

By default, all triggers are automatically enabled when Theyare created. However, they can be disabled. Once a task is completed, the trigger can be re–enabled. Enabling and Disabling triggers are performed using enable and disable keywords.

Syntax:

ALTER TRIGGER <triggername> ENABLE|DISABLE; Example:

ALTER TRIGGER empdel DISABLE;

For a particular table, all triggers can be enabled using a single command.

ALTER TABLE <tablename> DISABLE ALL TRIGGERS;

Dropping Triggers

Triggers are dropped using the Drop Trigger Command. The syntax follows :

Syntax:

RDBMS Concepts and Oracle 8i

346

DROP TRIGGER <triggername>;

Example:

DROP TRIGGER chkprod;

❑ **Check Your Progress – 3 :**

1. By default, all triggers are automatically ________ when they are created.

   a. enabled b. disabled

11.6 Package Creation :

A package is a schema object that groups logically related PL/SQL types, items, and supprograms. Packages usually have two parts, a specification and a body, although sometimes the body is unnecessary. The specification is the interface to your applications; it declares the types, variables and constants exceptions, cursors, and supprograms available for use. The body fully defines cursors and supprograms, and so implements the specification. A package once compiled and stored, gets life in a session when any of the components of the package is referred in the session. Then onwards, the variables, arrays, cursors or supprograms defined in the package are allocated and loaded in the memory for that session and any change that is done to the components are live for the whole session across operations and across all the supprograms and triggers. This is unlike a supprogram where the variables have effect only inside the supprogram and not across programs in the session.

Unlike supprograms, packages cannot be called, parameterized, or nested. Still, the format of a package is similar to that of a supprogram as shown below. CREATE PACKAGE name AS -- specification (visible part)

```
-- public type and item declarations
-- supprogram specifications
END [name];
CREATE PACKAGE BODY name AS -- body (hidden part)
-- private type and item declarations
-- supprogram bodies
[BEGIN
-- initialization statements]
END [name];
```

The specification holds public declarations, which are visible to the application. The body holds implementation details and private declarations, which are hidden from the application. This concept is very useful when modular applications are build which would enable a server centric application with all the application related programs and variables stored as part of the package.

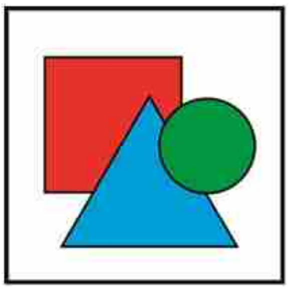


Fig. 11.1 Pakage Creation

You can debug, enhance, or replace a package body without changing the interface (package specification) to the package body.

To create packages and store them permanently in an Oracle database, CREATE PACKAGE and CREATE PACKAGE BODY statements are used. These statements can be executed interactively from SQL*Plus or Enterprise Manager.

In the example given below, you package a record type, a cursor, and two employment procedures are packaged. Notice that the procedure hire_employee uses the database sequence empno_seq and the function SYSDATE to insert a new employee number and hire date, respectively.

```
CREATE PACKAGE emp_actions AS – specification
PROCEDURE hire_employee (
ename VARCHAR2,
job VARCHAR2,
    mgr NUMBER,
    sal NUMBER,
    comm NUMBER,
    deptno NUMBER);
    PROCEDURE fire_employee (emp_id NUMBER);
    END emp_actions;
    CREATE PACKAGE BODY emp_actions AS -- body
    PROCEDURE hire_employee (
    ename VARCHAR2,
    job VARCHAR2,
    mgr NUMBER,
    sal NUMBER,
    comm NUMBER,
    deptno NUMBER) IS
    BEGIN
    INSERT INTO emp VALUES (empno_seq.NEXTVAL, ename, job,
    mgr, SYSDATE, sal, comm, deptno);
    END hire_employee;
    PROCEDURE fire_employee (emp_id NUMBER) IS
    BEGIN
    DELETE FROM emp WHERE empno = emp_id;

    END fire_employee;
END emp_actions;
```

This package creates two procedures called hire_employee and fire_employee. The two procedures can be executed in the following way :

```
EXEC
EMP_ACTIONS.HIRE_EMPLOYEE('SUDHA','MANAGER',7902,10000,200,10);
EXEC EMP_ACTIONS.FIRE_EMPLOYEE(2);
```

Only the declarations in the package specification are visible and accessible to applications.

Implementation details in the package body are hidden and inaccessible. So, the body (implementation) can be changed without having to recompile the calling programs.

Advantages of Packages

Packages offer several advantages: modularity, easier application design, information hiding, added functionality, and better performance.

Modularity

Packages logically related types, items, and supprograms in a named PL/SQL module to be encapsulated. Each package is easy to understand, and the interfaces between packages are simple, clear, and well defined. This aids application development.

Easier Application Design

When designing an application, all that is needed initially is the interface information in the package specifications. A specification can be coded and compiled without its body. Then, stored supprograms that reference the package can be compiled as well. The package bodies fully until the user is ready to complete the application.

Information Hiding

With packages, you can specify which types, items, and supprograms are public (visible and accessible) or private (hidden and inaccessible). For example, if a package contains four supprograms, three might be public and one private. The package hides the definition of the private supprogram so that only the package (not your application) is affected if the definition changes. This simplifies maintenance and enhancement. Also, by hiding implementation details from users, you protect the integrity of the package.

Added Functionality

Packaged public variables and cursors persist for the duration of a session. So, they can be shared by all supprograms that execute in the environment. Also, they allow data to be maintained across transactions without having to store it in the database.

Better Performance

When a packaged supprogram is called for the first time, the whole package is loaded into memory. So, later calls to related supprograms in the package require no disk I/O. Also, packages stop cascading dependencies and so avoid unnecessary recompiling. For example, if the definition of a packaged function is changed. Oracle need not recompile the calling supprograms because they do not depend on the package body.

❑ **Check Your Progress – 4 :**

1. A ________ is a schema object that groups logically related PL/SQL types, items, and supprograms.

   a. Package b. Program

2. The specification holds public declarations, which are visible to the ________.

   a. Application b. Package

3. To create packages and store them permanently in an Oracle database, ________ and CREATE PACKAGE BODY statements are used.

   a. CREATE PACKAGE b. TART PACKAGE

4. ________ details in the package body are hidden and inaccessible.

   a. Implementation b. Application

5. ________ logically related types, items, and supprograms in a named PL/SQL module to be encapsulated.

   a. Packages b. Program

11.7 Package Subprogram :

A supprogram is a program unit/module that performs a particular task. These supprograms are combined to form larger programs. This is basically called the 'Modular design'. A supprogram can be invoked by another supprogram or program which is called the calling program.

A supprogram can be created:

- At schema level
- Inside a package
- Inside a PL/SQL block

A schema level supprogram is a standalone supprogram. It is created with the CREATE PROCEDURE or CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

A supprogram created inside a package is a packaged supprogram. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement. We will discuss packages in the chapter 'PL/SQL – Packages'.

PL/SQL supprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of supprograms:

- **Functions :** these supprograms return a single value, mainly used to compute and return a value.
- **Procedures :** these supprograms do not return a value directly, mainly used to perform an action.

❑ **Check Your Progress – 5 :**

1. A ________ is a program unit/module that performs a particular task.

   a. Supprogram b. Sub module

2. A supprogram can be invoked by another supprogram or program which is called the ________.

   a. calling program b. second program

3. A ________ level supprogram is a stand alone supprogram.

   a. Schema b. Second

4. PL/SQL supprograms are named PL/SQL ________ that can be invoked with a set of parameters.

   a. Blocks b. Programs

5. PL/SQL provides ________ kinds of supprograms.

   a. Two b. Four

11.8 Let Us Sum Up :

In this chapter, detailed discussion was made PL/SQL procedures, supprograms, and the structures associated with this procedural language extension to the Oracle database. You also saw a sample SQL script for building a packaged procedure.

Triggers are set of supprograms that are automatically executed whenever a DML statement is issued on a particular table. They are used to automatically generate values, prevent invalid transactions and so on.

Triggers contain 3 parts :

- Triggering Event
- Restriction
- Action

There are 12 types of triggers.

Triggers are created using CREATE TRIGGER statement.

Instead of Triggers are used for Views which contain Joins.

Triggers can be enabled, disabled or dropped.

In this chapter, detailed discussion was made PL/SQL procedures, supprograms, and the structures associated with this procedural language extension to the Oracle database. You also saw a sample SQL script for building a packaged procedure.

11.9 Answers for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a **2 :** a **3 :** a **4 :** a **5 :** a

❑ **Check Your Progress 2 :**

1 : a **2 :** a **3 :** a **4 :** a

❑ **Check Your Progress 3 :**

1 : a

❑ **Check Your Progress 4 :**

1 : a **2 :** a **3 :** a **4 :** a **5 :** a

❑ **Check Your Progress 5 :**

1 : a **2 :** a **3 :** a **4 :** a **5 :** a

11.10 Glossary :

Triggers : They are set of supprograms that are automatically executed.

11.11 Assignment :

1. What do you understand by database triggers ?
2. Discuss the purpose of database triggers.
3. Discuss Package creation.
4. Write a note on package supprograms

11.12 Activities :

1. Discuss the parts of triggers.
2. Discuss the before and after option in triggers.

11.13 Case Study :

Discuss the process of creating triggers.

11.14 Further Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw–Hill Publication.
3. Fundamentals of Database Systems by Ramez Elmsari, Shamkant B Navathe – Pearson Education
4. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
5. Database Management System by Raghu Ramkrishnan – Tata McGraw–Hill Publication.
6. SQL, PL/SQL: The Programming Language Oracle – Ivan Bayross – BPB Publication.

BLOCK SUMMARY :

PL/SQL programming is the topic of this chapter. The subject areas of PL/SQL that were discussed include overview of PL/SQL, modular coding practices, developing PL/SQL blocks, interacting with Oracle, controlling process flow with conditional statements and loops, cursors, and error handling. PL/SQL is the best method available for writing and managing stored procedures that work with Oracle data. PL/SQL code consists of three su?blocks–the declaration section, the executable section, and the exception handler. In addition, PL/SQL can be used in four different programming constructs. The types are procedures and functions, packages, and triggers. Procedures and functions are similar in that they both contain a series of instructions that PL/SQL will execute. However, the main difference is that a function will always return one and only one value.

In this block, we have learnt about Store procedure. This portion provided an overview of the above topic with the help of various examples.

This block gave the student's a basic idea of what store procedure are. Various functions and commands have been discussed here in this block in a very brief and interesting way. The writer tried his best to concise the topics using the most easy language to help the student to understand the topic. The aim of this block was to enable the reader understand the basic concepts of these topcis. The writer tried his best to detail the store procedure have even been discussed by him in detail.

The objective of this block includes detailing the student's about the various above mentioned topics in very detail with the help of sufficient and suitable examples.

BLOCK ASSIGNMENT :

❖ **Short Questions :**

Write short notes on :

1. Variables
2. Constants
3. Data types and Arrays
4. Control Statements
5. Overloading
6. Commit statement
7. Package Creation

❖ **Long Questions :**

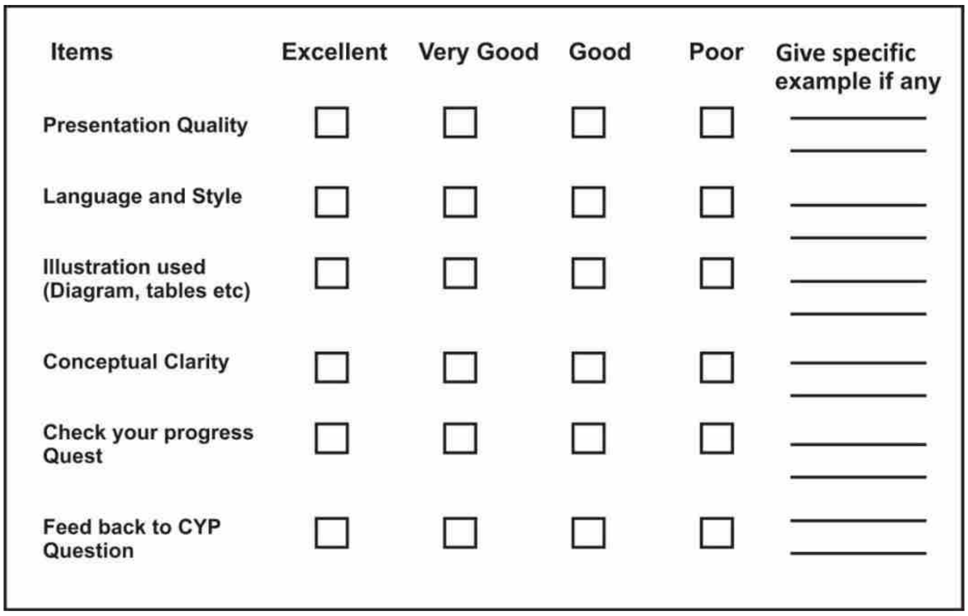
1. Discuss the control statements in detail with example.
2. What do you understand by internal functions ?
3. Discuss the naming conventions.
4. What do you understand by package subprograms ?

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	8	9	10	11
No. of Hrs.				

2. Please give your reactions to the following items based on your reading of the block :



Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	
Language and Style	☐	☐	☐	☐	
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	
Conceptual Clarity	☐	☐	☐	☐	
Check your progress Quest	☐	☐	☐	☐	
Feed back to CYP Question	☐	☐	☐	☐	

3. Any other Comments

....

....

....

....

....

....

....

....

Dr. Babasaheb Ambedkar Open University Ahmedabad **BCAR 302**

RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)

BLOCK 4 : PL/SQL

PL/SQL

Block Introduction :

This section covers the PL/SQL constructs that allow software developers to use this procedural language interface to the Oracle RDBMS. Constructs in PL/SQL are analogous to those in 3GL and C, and they allow a flexible technique of programming. To alter data as SQL*Forms relies heavily on PL/SQL, but You can access common datas processes by writing database procedures outside of forms. While limiting the amount of manipulation procedures and other Oracle developer tools applications in the form. The purpose of this lock is to help the reader grasp the fundamental ideas of the importance of PL/SQL. The author has done his best to describe the package's creation in detail. Apart from that, the variables and constants have been discussed in suprograms detail. The reader will also gain a comprehensive understanding of data types and arrays. On the other hand, the author has gone over the declaration statement and naming. Apart from that, other PL/SQL illustrations and examples have been provided.

Block Objectives :

After learning this block, you will be able to understand :

- The Package Creation
- Utility of Package Subprograms
- About the Variables and Constants
- About various Data types and Arrays
- Control Statements,
- DECLARE statement and Naming Conventions
- PL/SQL more with the help of examples and illustrations.
- Cursor Management and Exception Handling in PL/SQL and Packages

Block Structure :

Unit 12

CURSOR MANAGEMENT AND EXCEPTION HANDLING IN PL/SQL AND PACKAGES

UNIT STRUCTURE

12.0 Learning Objectives :

After learning this unit, you will be able to understand :

- The Errors
- Types of Cursors
- About Cursor Variables
- About eg. For using constrained cursors

12.1 Introduction :

This chapter deals with working of Cursors and different types of Cursors. This chapter also gives an insight into the working of composite data types and their methods. This chapter also introduces Cursor variables and their usage.

- Cursors
- Implicit Cursors
- Explicit Cursors
- Cursor Attributes
- Constrained Cursors and Unconstrained Cursors
- Using Cursor For update
- Using Composite Data types

Oracle uses work areas to execute SQL statements and store processing results. A PL/SQL construct called Cursor allows a work area to be named, to store and accesses its information. They are typically used in areas where the query inside the PL/SQL block retrieves more than one record. A cursor is a pointer to handle the context area or the memory area or work area. The result set of information is called Result Set.

12.2 Cursors :

A cursor is a data structure that stored procedures and Pre–processor 2 use at runtime to point to the result rows in a response set returned by an SQL query.

Stored procedures and embedded SQL also use cursors to manage inserts, updates, execution of multi statement requests, and SQL macros.

The syntax of functionally similar cursor control statements sometimes varies depending on whether they are used for stored procedures or embedded SQL. This book documents variant syntax forms with the individual cursor control statements. Several cursor control statements are valid only with embedded SQL. Cursors are not valid for sessions you conduct interactively from a terminal using a query manager like BTEQ.

Why Cursors Are Necessary ?

This information does not apply to result set cursors.

An embedded or stored procedure SQL SELECT statement can retrieve at most one row of dataat a time. It is an error for a SELECT statement to retrieve more than one row of data in these applications.

Without knowing the number of rows to be retrieved from a request, it is impossible to know the number of host variables required to hold the results of the SELECT. Thus, only a single result row is allowed.

This is not a problem for so–called singleton SELECTs, which are SELECT statements that you write so that they return only one row. However, SQL queries frequently return multiple rows in the form of a result table or response set. This situation is one that typical programming languages are not equipped to handle.

Traditional programming languages such as COBOL, C, and PL/I are record–oriented, while relational databases and their operators are inherently set–oriented.

Cursors enable record–oriented languages to process set–oriented data. Think of a cursor as a pointer to a single data row in a result table.

Cursors use SQL statements unique to embedded SQL and stored procedures to step through the result table, which is held in a data structure known as a spool file, one row at a time.

Types of Cursors :

PL/SQL implicitly declares a cursor for all SQL data manipulation statements, including queries that return only one row. Cursors can be of 2 types.

- Implicit Cursors
- Explicit Cursors

Implicit Cursors :

Whenever a SQL statement is issued the Database server opens an area of memory in which the command is parsed and executed. This area is called a cursor. When the executable part of PL/SQL block issues a SQL command, PL/SQL creates an implicit cursor, which has the identifier SQL.PL/SQL manages this cursor.

Explicit Cursors :

SELECT statements that occur within PL/SQL blocks are known as embedded. They must return one row and may only return one row. To get around this a SELECT statement is defined as a cursor (an area of memory), the query is executed and the returned rows are manipulated within the cursor. Explicit Cursors can be of two types.

- Static Cursors
- Dynamic Cursors

Static Cursors :

Static cursors are a type of cursors where the SELECT statement is given at compile time itself. That is, the table from which the data are coming and the records that are going to be selected are predetermined at compile time itself. The definition of the cursor is done in the declarative part.

Dynamic Cursors :

Dynamic Cursors as the name suggests, are a set of cursors where the records from the tables are selected at run time rather than at compile time.

Each Cursor has Four Attributes

%ROWCOUNT	Returns the number of rows processed by a SQL statement.
%FOUND	Holds TRUE if at least one row is processed
%NOTFOUND	Holds TRUE if no rows are processed.
%ISOPEN	Holds TRUE if a cursor is open or FALSE if cursor has not been opened or has been closed. They are used only in connection with explicit cursors.
%ROWCOUNT	The %ROWCOUNT attribute is used to return the number of records fetched. This is in accordance with the number of FETCHes done.
%FOUND	This attribute contains TRUE if the FETCH statement fetches any records. The attribute will hold FALSE if there are no records to be fetched.
%NOTFOUND	It is the logical opposite of % FOUND. If records are fetched, the %NOTFOUND is FALSE and TRUE if there are no more records to be processed. Using this we can terminate from the loop.
%ISOPEN	This attribute checks for the status, if the status of the cursor is open or not. If the cursor is opened, it holds TRUE and FALSE if the cursor is not opened.

The tabular structure illustrates this better.

Attribute	Is TRUE	Is FALSE
%ISOPEN %FOUND	If the cursor is opened. When records are fetched	If the cursor is not open. If there are no more
%NOTFOUND	using FETCH statement When there are no records available to be fetched	records to be fetched and processed When records are fetched by the FETCH statement
%ROWCOUNT	Holds the number of records	–

Explicit Cursors :

Explicit cursor manipulation is performed using four commands

- DECLARE
- OPEN
- FETCH
- CLOSE

❑ **Check Your Progress – 1 :**

1. A ________ is a data structure that stored procedures and Preprocessor 2 use at runtime to point to the result rows in a response set returned by an SQL query.

   a. Cursor b. pointer

2. ________ and embedded SQL also use cursors to manage inserts, updates, execution of multi statement requests, and SQL macros.

   a. Stored procedures b. Statements

3. The syntax of functionally similar cursor control statements sometimes varies depending on whether they are used for ________ or embedded SQL.

   a. stored procedures b. Cursor

4. Several ________ control statements are valid only with embedded SQL.

   a. Cursor b. Pointer

5. Cursors enable ________ languages to process set–oriented data.

   a. record–oriented b. Program oriented

12.3 Explicit Cursor Attributes :

There are four attributes associated with cursors : ISOPEN, FOUND, NOTFOUND, and ROWCOUNT. These attributes can be accessed with the % delimiter to obtain information about the state of the cursor. The syntax for a cursor attribute is : cursor_name%attribute where cursor_name is the name of the explicit cursor. The behaviors of the explicit cursor attributes are described in the following table.

Attribute	Description
%ISOPEN	TRUE if cursor is open. FALSE if cursor is not open.
%FOUND	INVALID_CURSOR is raised if cursor has not been OPENed. NULL before the first fetch. TRUE if record was fetched successfully. FALSE if no row was returned. INVALID_CURSOR if cursor has been CLOSEd.
%NOTFOUND	INVALID_CURSOR is raised if cursor has not been OPENed. NULL before the first fetch. FALSE if record was fetched successfully. TRUE if no row was returned. INVALID_CURSOR if cursor has been CLOSEd.
%ROWCOUNT	INVALID_CURSOR is raised if cursor has not been OPENed. The number of rows fetched from the cursor. INVALID_CURSOR if cursor has been CLOSEd.

Frequently a cursor attribute is checked as part of a WHILE loop that fetches rows from a cursor:

```
        DECLARE
        caller_reccaller_pkg.caller_cur%ROWTYPE;
        BEGIN
        OPEN caller_pkg.caller_cur;
        LOOP
        FETCH caller_pkg.caller_cur into caller_rec;
          EXIT WHEN caller_pkg.caller_cur%NOTFOUND
                         OR
          caller_pkg.caller_cur%ROWCOUNT> 10;
          UPDATE call
          SET caller_id = caller_rec.caller_id
          WHERE call_timestamp< SYSDATE;
          END LOOP;
          CLOSE caller_pkg.caller_cur;
   END;
```

%ISOPEN Attribute

You can fetch rows only when the cursor is open. Use the %ISOPEN cursor attribute before performing a fetch to test whether the cursor is open. %ISOPEN returns the status of the cursor: TRUE if open and FALSE if not.

Example :

```
IF NOT emp_cursor%ISOPEN THEN
OPEN emp_cursor;
END IF;
LOOP
FETCH emp_cursor...
```

%ROWCOUNT and %NOTFOUND Attributes

Usually the %ROWCOUNT and %NOTFOUND attributes are used in a loop to determine when to exit the loop. Use the %ROWCOUNT cursor attribute for the following:

- To process an exact number of rows
- To count the number of rows fetched so far in a loop and/or determine when to exit the loop

Use the %NOTFOUND cursor attribute for the following:

- To determine whether the query found any rows matching your criteria
- To determine when to exit the loop

Example of %ROWCOUNT and %NOTFOUND

This example shows how you can use %ROWCOUNT and %NOTFOUND attributes for exit conditions in a loop.

```
DECLARE
CURSOR emp_cursor IS
SELECT employee_id, last_name FROM employees;
v_emp_recordemp_ cursor%ROWTYPE; BEGIN

OPEN emp_cursor;
LOOP
FETCH emp_cursor INTO v_emp_record;
EXIT WHEN emp_cursor%ROWCOUNT> 10 OR emp_cursor%
NOTFOUND;
DBMS_OUTPUT.PUT_LINE(v_emp_record.employee_id ||' ?||
v_emp_record.last_name);

END LOOP;
CLOSE emp_cursor;

END;
```

Explicit Cursor Attributes in SQL Statements

You cannot use an explicit cursor attribute directly in an SQL statement. The following code returns an error:

```
DECLARE
CURSOR emp_cursor IS
SELECT employee_id, salary FROM employees
ORDER BY SALARY DESC;
v_emp_recordemp_cursor%ROWTYPE;
v_count NUMBER;
BEGIN
OPEN emp_cursor;
LOOP

FETCH emp_cursor INTO v_emp_record;
EXIT WHEN emp_cursor%NOTFOUND;
INSERT INTO top_paid_emps
(employee_id, rank, salary)
VALUES
(v_emp_record.employee_id, emp_cursor%ROWCOUNT,
v_emp_record.salary);
```

Usage Notes :

The cursor attributes apply to every cursor or cursor variable. For example, you can open multiple cursors, then use %FOUND or %NOTFOUND to tell which cursors have rows left to fetch. Likewise, you can use %ROWCOUNT to tell how many rows have been fetched so far.

If a cursor or cursor variable is not open, referencing it with %FOUND, %NOTFOUND, or %ROWCOUNT raises the predefined exception INVALID_ CURSOR.

When a cursor or cursor variable is opened, the rows that satisfy the associated query are identified and form the result set. Rows are fetched from the result set one at a time.

If a SELECT INTO statement returns more than one row, PL/SQL raises the predefined exception TOO_MANY_ROWS and sets %ROWCOUNT to 1, not the actual number of rows that satisfy the query.

Before the first fetch, %NOTFOUND evaluates to NULL. If FETCH never executes successfully, the EXIT WHEN condition is never TRUE and the loop is never exited. To be safe, you might want to use the following EXIT statement instead:

```
EXIT WHEN c1%NOTFOUND OR c1%NOTFOUND IS NULL;
```

You can use the cursor attributes in procedural statements but not in SQL statements.

❑ **Check Your Progress – 2 :**

1. There are ________ attributes associated with cursors.

   a. Four b. Five

2. You can fetch rows only when the cursor is ________.

   a. Open b. closed

3. Usually the %ROWCOUNT and %NOTFOUND attributes are used in a loop to determine when to ________ the loop.

   a. Exit b. Enter

4. The cursor attributes apply to ________ cursor or cursor variable.

   a. Every b. unique

5. When a cursor or ________ variable is opened, the rows that satisfy the associated query are identified and form the result set.

   a. Cursor b. Fixed

12.4 Creation of Cursor Packages :

A cursor is an image that tracks the mouse on the display. Each window has its own cursor which you can change. There are some cursors defined by OPEN LOOK that correspond to specific window manager operations such as resizing or dragging windows. For these cases, you cannot redefine a cursor. However, for windows in your application, you can assign any cursor image you like.

To use the CURSOR package, include the header file <xview/cursor.h>. The owner of the cursor may be any XView object. The root window associated with the XView object is used internally by the CURSOR package. If the owner is NULL, then the root window of the default screen is used.

A number of predefined cursors are available in the CURSOR package for use as OPEN LOOK cursors. To use these cursors, you may specify the CURSOR_SRC_CHAR and CURSOR_ MASK_CHAR attributes with certain predefined constants as values for these attributes. There are some OPEN LOOK cursor defines prefixed by OLC_ in <xview/cursor.h>.

The hotspot on a cursor is the location in which the cursor is located if the user generates an event like pressing a mouse button or typing at the keyboard, or if you were to query its position. For example, if a cursor is shaped like an arrow, the hotspot should be at the tip of the arrow. If the hotspot for a cursor were set to (0, 0) then the hotspot would be the upper–left corner of the image used. A cursor shaped like a?ull's eye (16x16) might have its hotspot at (7, 7) to indicate that the focus for the cursor is in the middle.

❑ **Check Your Progress – 3 :**

1. A cursor is an ________ that tracks the mouse on the display.

   a. Image b. Picture

2. To use the ________ package, include the header file <xview/cursor.h>.

   a. CURSOR b. Image

3. A number of predefined cursors are available in the CURSOR package for use as ________ cursors.

   a. OPEN LOOK   b. LOOK

4. The hotspot on a cursor is the location in which the cursor is located if the ________ generates an event like pressing a mouse button or typing at the keyboard, or if you were to query its position.

   a. User   b. System

12.5 Let Us Sum Up :

In this unit, we have learnt that Cursors are named workareas that are used for multiple rows processing in PL/SQL blocks.

They can be of two types:

- Implicit Cursors and Explicit Cursors
- The four Cursor attributes which check the status of a cursor are %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.

Composite Datatypes are the datatypes that can hold multiple columns and multiple records.

Dynamic cursors can accept the table name at run–time. They can be further classified as Constrained and Un–constrained Cursors. Cursor variables can be associated with different statements at run time

12.6 Answers for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a   **2 :** a   **3 :** a   **4 :** a   **5 :** a

❑ **Check Your Progress 2 :**

1 : a   **2 :** a   **3 :** a   **4 :** a   **5 :** a

❑ **Check Your Progress 3 :**

1 : a   **2 :** a   **3 :** a   **4 :** a

12.7 Glossary :

Cursor : They are a data structure that stored procedures and reprocessor 2 use at runtime to point to the result rows in a response set returned by an SQL query.

12.8 Assignment :

1. What do you understand by cursors ? Why are they considered necessary ?
2. Discuss the types of cursors.

12.9 Activities :

Explain working of cursors in real time applications

12.10 Case Study :

Make a case study on applications where cursors can be used

12.11 Further Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw– Hill Publication.
3. An Introduction to Database Systems – Bipin Desai– Galgotia Publication.
4. DatabaseManagement System by Raghu Ramkrishnan– Tata McGraw – Hill Publication.
5. SQL, PL/SQL: The Programming Language Oracle – Ivan Bayross – BPB Publication.

Unit 13

PACKAGE AND EXCEPTIONS

UNIT STRUCTURE

13.0 Learning Objectives :

After learning this unit, you will be able to understand :

- Exceptions
- More details on Exceptions

13.1 Introduction :

Runtime errors arise from design faults, mistakes, hardware failures and many other sources. Although these errors are not anticipated, these errors can be handled meaningfully. To capture the errors raised, Exceptions are used.

13.2 Exception Handling :

In PL/SQL, warnings or error condition is called Exception. When an error occurs, an exception is raised. That is normal execution is stopped and control is transferred to the exception–handling part of the PL/SQL block. They are designed for run–time rather than compile time errors. They are handled in the Exception section of the PL/SQL block. Errors can be classified as

- Runtime Errors
- Compile–time Errors

Exceptions can be broadly classified into :

- Predefined Exceptions
- User–defined Exceptions
- Un–defined Exceptions

Pre–defined Exceptions

Predefined exceptions are exceptions that are already defined by Oracle. The following list gives the set of Predefined Exceptions.

Advantages of Exceptions

Using exceptions for error handling has several advantages. Without exception handling, every time a command is issued, execution errors must be checked, as follows:

```
BEGIN


SELECT ...


--    check for ‗no data found' error SELECT
...
--    check for ‗no data found' error SELECT
...
--    check for ‗no data found' error
```

Error processing is not clearly separated from normal processing; nor is it robust. If you neglect to code a check, the error goes undetected and is likely to cause other, seemingly unrelated errors. With exceptions, errors can be conveniently handled without the need to code multiple checks, as follows:

```
BEGIN
SELECT ...
SELECT ...
SELECT ...
...
Exception
WHEN NO_DATA_FOUND THEN -- catches all ‗no data found' errors
```

Exceptions improve readability by letting error–handling routines to be isolated. Error recovery algorithms do not obscure the primary algorithm. Exceptions also improve reliability. You need not worry about checking for an error at every point it might occur. Just add an exception handler to your PL/SQL block. If the exception is ever raised in that block (or any sub–block), you can be sure it will be handled.

STORAGE_ERROR	PL/SQL runs out of memory or memory is corrupted.
SUBSCRIPT_BEYON	A nested table or varray element is referenced using an index number
D_COUNT	larger than the number of elements in the collection.
SUBSCRIPT_OUTSIDE_LIMIT	you reference a nested table or varray element using an index number that is outside the legal range (–1 for example).
TIMEOUT_ON_RESOURCE	A timeout occurs while Oracle is waiting for a resource.

TOO_MANY_ROWS	A SELECT INTO statement returns more than one row. Anarithmetic conversion, truncation, or size–constraint error occurs. For example, when a column value is selected into a character variable, if the value is longer than the declared length of the variable,
VALUE_ERROR	PL/SQL aborts the assignment and raises VALUE_ERROR. In procedural statements, VALUE_ERROR is raised if the conversion of a character string to a number fails. In SQL statements, INVALID_NUMBER is raised.
ZERO_DIVIDE	User tries to divide a number by zero.

The following examples illustrates the usage of Pre–defined exceptions Zero–Divide When a number is divided by zero, this exception is raised. The following example briefs this exception.

Example

```
DECLARE

X NUMBER:=&X;

Y NUMBER:=&Y;

BEGIN

DBMS_OUTPUT.PUT_LINE(‗RESULT IS ‗||X/Y); END;
```

In this example, two values are accepted for x and y respectively. If both contains non–zero values the result is printed. If ‗y' contains zero, it leads to the exception which is displayed as :

ORA–01476: divisor is equal to zero

In order to handle the exception, exception clause must contain the Exception Zero_Divide.

Refining the above example,

```
DECLARE

X NUMBER:=&X;

Y NUMBER:=&Y;

BEGIN

DBMS_OUTPUT.PUT_LINE(‗RESULT IS ‗||X/Y);
```

```
EXCEPTION

WHEN ZERO_DIVIDE THEN

DBMS_OUTPUT.PUT_LINE(?VALUE IS DIVIDED BY ZERO');
END;
```

The second example illustrates the usage of NO_DATA_FOUND

Example

```
DECLARE

MYENAME VARCHAR2(20);

BEGIN

SELECT ENAME INTO MYENAME FROM EMP WHERE
EMPNO=&X;

DBMS_OUTPUT.PUT_LINE(‗THE NAME OF THE EMPLOYEE IS
‗||MYENAME);

EXCEPTION

WHEN NO_DATA_FOUND THEN

DBMS_OUTPUT.PUT_LINE(?NO RECORD FOUND');

END;

/
```

A SELECT statement returns not more than one row. If a SELECT statement is made to return more than one record, a cursor is required. The following example shows this.

Example

```
DECLARE

MYSAL NUMBER;

BEGIN

SELECT SAL INTO MYSAL FROM EMP WHERE DEPTNO=10;

DBMS_OUTPUT.PUT_LINE(MYSAL);

END;
```

This would display

ORA–01422: exact fetch returns more than requested number of rows

Raises the pre–defined exception too_many_rows. To avoid this, handle the exception in the Exception clause.

```
DECLARE

MYSAL NUMBER;

BEGIN

SELECT SAL INTO MYSAL FROM EMP WHERE DEPTNO=10;

DBMS_OUTPUT.PUT_LINE(MYSAL);

EXCEPTION

WHEN TOO_MANY_ROWS THEN

DBMS_OUTPUT.PUT_LINE(‗Use Cursors');

END;
```

The given above examples illustrated the usage of handling pre–defined exceptions in the exception section of any PL/SQL block. Likewise all the other pre–defined exceptions can behandled in the following fashion.

Using Others

To handle any kind of an exception, use OTHERS. This is a handler that handles anyexception. The example follows:

Example

```
DECLARE

MYSAL NUMBER;

MYENAME VARCHAR2(20);

BEGIN

SELECT SAL INTO MYSAL FROM EMP WHERE DEPTNO=10;

DBMS_OUTPUT.PUT_LINE(MYSAL);

SELECT ENAME INTO MYENAME FROM EMP WHERE
EMPNO=&X;

DBMS_OUTPUT.PUT_LINE(‗THE NAME OF THE EMPLOYEE
IS ‗||MYENAME);

EXCEPTION

WHEN TOO_MANY_ROWS THEN
```

Exception	Raised when ...
ACCESS_INTO_NULL	User tries to assign values to the attributes of an uninitialized (atomically null) object. User tries to apply collection methods other than EXISTS to an uninitialized (atomically null) nested table or varray, or assigns values to the elements of an uninitialized nested table or varray.
COLLECTION_IS_NULL	User tries to open an already open cursor. A cursor must be Closed before it can be reopened.
CURSOR_ALREADY_OPEN	A cursor FOR loop automatically opens the cursor to which it refers. So, that cursor cannot be opened inside the loop.
DUP_VAL_ON_INDEX	User tries to store duplicate values in a database column that is constrained by a unique index.
INVALID_CURSOR	User tries an illegal cursor operation such as closing an unopened cursor.
INVALID_NUMBER	In a SQL statement, the conversion of character string to a number fails because the character string does not represent a valid number. In procedural statements, VALUE_ERROR is raised.
LOGIN_DENIED	User tries logging on to Oracle with an invalid username and/or password.
NO_DATA_FOUND	A SELECT INTO statement returns no rows, or a Deleted element is referenced in a nested table, or an Uninitialized element is referenced in an index–by table. The FETCH statement is expected to return no rows eventually, so when that happens, no exception is raised. SQL group functions such as AVG and SUM always return a value or a null. So, a SELECT INTO statement that calls a group function will never raise NO_DATA_FOUND

```
DBMS_OUTPUT.PUT_LINE(‗Use Cursors');

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE(‗OTHERS EXCEPTION HANDLED');

END;
```

❑ **Check Your Progress – 1 :**

1. ________ errors arise from design faults, mistakes, hardware failures and many other sources.

   a. Runtime b. Exceptions

2. To capture the errors raised, ________ are used

   a. Exceptions b. runtime

3. ________ exceptions are exceptions that are already defined by Oracle.

   a. Predefined b. defined

4. Using exceptions for error handling has several ________.

   a. Advantages b. disadvantages

5. ________ improve readability by letting error–handling routines to be isolated.

   a. Exceptions b. Runtime errors

13.3 More on Exceptions :

User–Defined Exceptions

PL/SQL allows users to define their own exceptions. Unlike Predefined exceptions, user defined exceptions must be declared exceptions can be raised using one of the following :

- RAISE
- RAISE_APPLICATION_ERROR

In the case of user–defined exceptions, the following three steps must be performed :

- Declare the exception
- Raise the Exception
- Handle the Exception

Declaring Exceptions

Exceptions are declared in the declarative part of a PL/SQL block, supprogram or package. It contains a name followed by the datatype called Exception. The following syntax shows this :

```
DECLARE

Ex1 EXCEPTION;
```

Exception declaration is very much similar to variable declaration. But they cannot appear as apart of an assignment statement.

Raising Exceptions

Raising Exceptions means the exception declared by the user is raised explicitly unlike Predefined exceptions. The keyword RAISE is used to perform this operation. Syntax forusing this is :

```
RAISE ex1;
```

Handling Exceptions

After the exception is raised, it must be handled. Handling of exceptions is performed using the exception handling section similar to handling of pre–defined exceptions.

```
WHEN ex1 THEN

……
```

The following example illustrates this:

Example

```
DECLARE

Ex1 EXCEPTION;

N NUMBER(5):=&x;

BEGIN

If N > 20000 THEN RAISE

ex1;

END IF; EXCEPTION

WHEN ex1 THEN

DBMS_OUTPUT.PUT_LINE(‗Value must not exceed 20000'); END;
```

Consider another example. The following example adds the accepted value onto the table. If the salary exceeds 30000 the exception SALARY_RAISE is raised and handled.

```
DECLARE

ENO NUMBER:=&NO;

NAME VARCHAR2(20):=‗&B';

SALARY NUMBER:=&P;

SALARY_RAISE EXCEPTION;

BEGIN

IF SALARY > 30000 THEN

RAISE SALARY_RAISE;

ELSE

INSERT INTO EMP(EMPNO,ENAME,SAL)
```

```
VALUES(ENO,NAME,SALARY);

END IF;

EXCEPTION

WHEN SALARY_RAISE THEN

DBMS_OUTPUT.PUT_LINE(‗SALARY EXCEEDS 30000');

END;
```

RAISE_APPLICATION_ERROR

Raise_Application_Error stops the program and terminates the block abruptly.

Raise_Application_error is primarily used in triggers to rollback the transaction and give suitable error messages. The syntax is

```
RAISE_APPLICATION_ERROR (errornumber , MESSAGE);
```

where the error messages span between –20000 and –20999. Message is a character data that can hold upto 2000 bytes. The following example illustrates this:

In the following example, raise_application_error is called if an employee's salary is missing:

```
DECLARE

Emp_idnumber :=&no;
current_salary NUMBER;

increase number:=&x;

BEGIN

SELECT sal INTO current_salary FROM emp

WHERE empno = emp_id;

IF current_salary IS NULL THEN

/* Issue user–defined error message. */

raise_application_error(–20101, ‗Salary is missing');

ELSE

UPDATE emp SET sal = current_salary + increase

WHERE empno = emp_id;
```

```
END IF;

EXCEPTION

WHEN NO_DATA_FOUND THEN

RAISE_APPLICATION_ERROR(–20000,'NO MATCHING RECORD');

END raise_salary;
```

This program accepts the employee number and the salary increase.If the corresponding employee is found, it updates the value else it raises an error. Assume 765 isa number which is not available in the table. The error message in this case wills be :

```
ERROR at line 1:

ORA–20000: NO MATCHING RECORD

ORA–06512: at line 17
```

Using EXCEPTION_INIT – Use of PRAGMA

To handle unnamed internal exceptions, you must use the OTHERS handler or the pragma EXCEPTION_INIT must be used. A pragma is a compiler directive, which can be thought of as a parenthetical remark to the compiler. Pragmas (also called pseudo instructions) are processed at compile time, not at run time.

In PL/SQL, the pragma EXCEPTION_INIT tells the compiler to associate an exception name with an Oracle error number. That allows any internal exception to be referenced by name and to write a specific handler for it.

PRAGMA EXCEPTION_INIT is given in the declarative part of a PL/SQL block, supprogram, or package using the syntax

```
PRAGMA EXCEPTION_INIT(exception_name, Oracle_error_number);
```

whereexception_name is the name of a previously declared exception. The pragma must appear somewhere after the exception declaration in the same declarative part, as shown in the following example:

```
DECLARE

X EXCEPTION;

PRAGMA EXCEPTION_INIT(X,–1400);

BEGIN

INSERT INTO EMP(EMPNO) VALUES(NULL);

EXCEPTION

WHEN X THEN
```

DBMS_OUTPUT.PUT_LINE(‗VALUE CANNOT HOLD NULL VALUES');

END;

In the above example, if a null value is being inserted onto a column which cannot contain null values, an exception is raised.

Unhandled Exceptions

Remember, if it cannot find a handler for a raised exception, PL/SQL returns an unhandled exception error to the host environment, which determines the outcome. For example, in the Oracle Precompilers environment, any database changes made by a failed SQL statement or PL/SQL block are rolled back.

Unhandled exceptions can also affect supprograms. If you exit a supprogram successfully,

PL/SQL assigns values to OUT parameters. However, if you exit with an unhandled exception, PL/SQL does not assign values to OUT parameters. Also, if a stored supprogram fails with an unhandled exception, PL/SQL does not roll back database work done by the supprogram.

Unhandled exceptions can be avoided by coding an OTHERS handler at the topmost level of every PL/SQL block and supprogram.

How Exceptions Propagate

When an exception is raised, if PL/SQL cannot find a handler for it in the current block or supprogram, the exception propagates. That is, the exception reproduces itself in successive enclosing blocks until a handler is found or there are no more blocks to search. In the latter case, PL/SQL returns an unhandled exception error to the host environment. An exception can propagate beyond its scope, that is, beyond the block in which it is declared. Consider the following example:

BEGIN

...

DECLARE ---------- sub–block begins

past_due EXCEPTION;

RDBMS Concepts and Oracle8i

293

BEGIN

...

IF ... THEN

RAISE past_due;

END IF;

```
END; ------------– sub–block ends

EXCEPTION

...

WHEN OTHERS THEN

ROLLBACK;

END;
```

Because the block in which it was declared has no handler for the exception named past_due, it propagates to the enclosing block. But, according to the scope rules, enclosing blocks cannot reference exceptions declared in a sub–block. So, only an OTHERS handler can catch the exception.

❑ **Check Your Progress – 2 :**

1. ________ allows users to define their own exceptions.

   a. PL/SQL b. Cursors

2. ________ are declared in the declarative part of a PL/SQL block, supprogram or package.

   a. Exceptions b. PL/SQL

3. Raising ________ means the exception declared by the user is raised explicitly unlike Predefined exceptions.

   a. Exceptions b. PL/SQL

4. After the ________ is raised, it must be handled.

   a. Exception b. PL/SQL

5. To handle unnamed ________ exceptions, you must use the OTHERS handler or the pragma.

   a. Internal b. External

13.4 Let Us Sum Up :

In this unit, we have learnt that Errors are of two types

- Runtime Errors
- Compile–time Errors

Exceptions are of three types

- Predefined Exceptions
- User–defined Exceptions
- Un–defined Exceptions

13.5 Answers for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a **2 :** a **3 :** a **4 :** a **5 :** a

❑ **Check Your Progress 2 :**

1 : a **2 :** a **3 :** a **4 :** a **5 :** a

13.6 Glossary :

Exceptions : In PL/SQL, warnings or error condition is called Exception. When an error occurs, an exception is raised

13.7 Assignment :

Discuss the types of Errors

13.8 Activities :

Describe the real time applications where exceptions can be useful

13.9 Case Study :

Discuss the case study where problems occurred due to lack of exception handling

13.10 Further Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth – Tata McGraw– Hill Publication.
3. An Introduction to Database Systems – Bipin Desai– Galgotia Publication.
4. DatabaseManagement System by Raghu Ramkrishnan– Tata McGraw – Hill Publication.
5. SQL, PL/SQL: The Programming Language Oracle – Ivan Bayross – BPB Publication.

Unit 14 PRACTICAL SOLVING OF SQL

UNIT STRUCTURE

14.0 Learning Objectives :

After learning this unit, you will be able to understand :

- The SQL language through practical excercises.
- The various SQL * formatting features.

14.1 Introduction :

The purpose of any exercise is to steadily develop skills and to acquire the automatic algorithms for fulfilling certain operations. As for the SQL language, practical exercises are intended to enable the database programmer to quickly devise SQL queries in order to solve practically any problem, by having already studied similar problems in exercises. This unit provides such a collection of practical solved SQL exercises.

This unit covers some common SQL* Plus report formatting features. It also covers techniques for controlling the resulting output. I discuss and provide examples of simple reporting techniques and advanced reporting techniques.

The following example formats the results of a SQL query. It defines a report title and formats, assigns column headings, and applies some control breaks for intermediate and report totalling.

14.2 Examples :

14.2.1 Example 1 – Simple SQL* Plus Report Code :

Listing 1. Simple SQL* Plus Report Code :

```
define ASSIGNED_ANALYST = &1
set FEEDBACK OFF
set VERIFY OFF
set TERMOUT OFF
set ECHO OFF
column APPLICATION_NAME format a12 heading 'Application'
column PROGRAM_NAME format a12 heading 'Program'
column PROGRAM_SIZE format 999999 heading 'Program|Size'
break on APPLICATION_NAME skip 2
break on report skip 2
compute sum of PROGRAM_SIZE on APPLICATION_NAME
compute sum of PROGRAM_SIZE on report
ttitle 'Programs by Application | assigned to: &ASSIGNED_ANALYST'
spool ANALYST.OUT
select APPLICATION_NAME,PROGRAM_NAME,nvl(PROGRAM_SIZE,0)
from APPLICATION_PROGRAMS
where ASSIGNED_NAME = '&&ASSIGNED_ANALYST'
order by APPLICATION_NAME,PROGRAM_NAME
/
spool off
exit
```

The following is the output report.

```
Tue Jul 13
Programs by Application
Assigned to :
Program
Application     Program       Size
------------    ------------  ---------
COBOL           CLAIMS        10156
HOMEOWN         22124
PREMIUMS        10345         HOTKA
---------
sum             42625
```

```
FORTRAN     ALGEBRA     6892
MATH1       7210
PL/SQL      SCIENCE1    10240
            ---------
            sum         24342
            sum         66967
```

Listing 1 is a simple but common form of SQL* Plus formatting. This report passes a command–line parameter (&1 on line 1) and assigns it to the variable name ASSIGNED_ANALYST. The ASSIGNED_ANALYST variable is then used in the headings (see line 13) and again as part of the SQL query (see line 17). Lines 2, 3, 4, and 5 suspend all terminal output from the SQL* Plus environment. The && is utilized to denote substitution of an already defined variable. This report contains two breaks, one when the column APPLICATION_NAME changes (see line 9) and one at the end of the report (see line 10). Totals are also calculated for each of these breaks (see lines 11 and 12). The pipe character (|) in the TTITLE command (see line 13) moves the following text onto its own line. Line 14 will open an operating system–dependent file named ANALYST.OUT in the current operating system–dependent directory. The order by clause of the query on line 18 ensures that the breaks occur in an orderly manner.

14.2.2 Example 2 – Cross–Tabular SQL* Plus Report Code :

Listing 2. Cross–tabular SQL* Plus Report Code :

```
1:   define RPT_DATE = &1
2:   set FEEDBACK OFF
3:   set VERIFY OFF
4:   set TERMOUT OFF
5:   set ECHO OFF
6:   column SALES_REP format a12 heading 'Sales|Person'
7:   column NISSAN format 999999 heading 'Nissan'
8:   column TOYOTA format 999999 heading 'Toyota'
9:   column GMformat 999999 heading 'GM'
10:  column FORD format 999999 heading 'Ford'
11:  column CRYSLER format 999999 heading 'Crysler'
12:  column TOTALS format 999999 heading 'Totals'
13:  break on report skip 2
14:  compute sum of NISSAN on report
15:  compute sum of TOYOTA on report
16:  compute sum of GM on report
17:  compute sum of FORD on report
18:  compute sum of CRYSLER on report
19:  compute sum of TOTALS on report
20:  ttitle left '&&IN_DATE' center 'Auto Sales' RIGHT 'Page: 'format
     999
```

```
21:  SQL.PNO skip CENTER ' by Sales Person '
22:  spool SALES.OUT
23:  select SALES_REP,
24:  sum(decode(CAR_TYPE,'N',TOTAL_SALES,0)) NISSAN,
25:  sum(decode(CAR_TYPE,'T',TOTAL_SALES,0)) TOYOTA,
26:  sum(decode(CAR_TYPE,'G',TOTAL_SALES,0)) GM,
27:  sum(decode(CAR_TYPE,'F',TOTAL_SALES,0)) FORD,
28:  sum(decode(CAR_TYPE,'C',TOTAL_SALES,0)) CRYSLER,
29:  sum(TOTAL_SALES) TOTALS
30:  from CAR_SALES
31:  where SALES_DATE <= to_date('&&RPT_DATE')
32:  group by SALES_REP
33:  /
34:  spool off
35:  exit
```

The following code shows the output

```
31–AUG–95      Auto Sales
by Sales Person
   Sales
  Person    Nissan  Toyota   GM    Ford  Crysler  Totals
Elizabeth    5500    2500     0      0    4500    12500
  Emily      4000    6000   4400   2000     0     16400
 Thomas      2000    1000   6000   4000   1500    14500
```

Listing 6.2 is a cross–tabular SQL* Plus command file. This report passes a command–line parameter (&1 on line 1) and assigns it to the variable name RPT_DATE. The RPT_DATE variable is then used in the headings (see line 20) and again as part of the SQL query (see line 31). Lines 2, 3, 4, and 5 suspend all terminal output from the SQL* Plus environment. The report will be created in the operating system–dependent file SALES.OUT. Column formatting commands control the appearance of the columns (lines 6 through 12). The combination of compute commands (lines 14 through 19), the sum statements in the query (lines 24 through 29), and the group by clause in the query (line 32) give the report output the appearance of a cross–tabular report.

14.2.3 Example 3 – Master/Detail SQL* Plus Report Code :

Listing 3. Master/detail SQL* Plus Report Code :

```
1:  ttitle 'Sales Detail | by Sales Rep'
2:  set HEADINGS OFF
3:  column DUMMY NOPRINT
4:  select 1 DUMMY, SALES_REP_NO,'Sales Person: ' || SALES_REP
5:  from sales
6:  UNION
```

```
select 2 DUMMY,SALES_REP_NO,'--------------------'
from sales
UNION
select 3 DUMMY,SALES_REP_NO, rpad(CAR_MAKE,4) || ' ' ||
to_char(SALE_AMT,'$999,999.99')
from sales_detail
UNION
select 4 DUMMY,SALES_REP_NO,' ----------'
from sales
UNION
select 5 DUMMY,SALES_REP_NO,'Total: ' ||
to_char(sum(TOTAL_SALES),'$999,999.99'))
from sales
UNION
select 6 DUMMY,SALES_REP_NO,' '
from sales
order by 2,1,3
/
```

I will now only include the specific SQL* Plus commands necessary to produce the desired output in the remaining examples.

The following code shows the output.

```
Thur Aug 31
                Sales Detail
by Sales Rep
Sales Person: Elizabeth
----------------------------
Chrysler    $3,000
Chrysler    $1,500
Nissan      $2,000
Nissan      $2,000
Nissan      $1,500
Toyota      $2,500
            ----------
Total:      $12,500
Sales Person: Emily
----------------------------
Ford        $1,000
Ford        $1,000
GM          $2,000
GM          $2,400
```

```
Nissan      $2,000
Nissan      $2,000
Toyota      $1,000
Toyota      $2,500
Toyota      $2,500
            ----------
Total:      $16,400
Sales Person: Thomas
----------------------------
Chrysler    $1,500
Ford        $1,000
Ford        $3,000
GM          $1,400
GM          $1,600
GM          $3,000
Nissan      $2,000
Toyota      $1,000
            ----------
Total:      $16,400
```

Listing 3 creates a master/detail SQL* Plus report by utilizing the SQL UNION command. In this example, there are six distinct separate types of lines to be printed: the sales person (line 4), a line of dashes under the sales person (line 7), the detail line (line 10), a line of dashes under the detail total (line 14), a total line (line 17), and a blank line (line 21). There are six separate queries that have their output merged and sorted together by the SQL JOIN statement (see lines 6, 9, 13, 16, 19, and 23). When you use JOIN to merge the output of two or more queries, the output result set must have the same number of columns. The headings are turned off (line 2) because regular SQL* Plus column headings are not desired for this type of report. The first column of each query has an alias column name of DUMMY. This DUMMY column is used to sort the order of the six types of lines (denoted by each of the six queries). The DUMMY column's only role is to maintain the order of the lines within the major sort field (SALES_REP_NO in this example); therefor, the NOPRINT option is specified in line 3.

Listing 4 uses the JOIN feature to display output from two or more tables within the same report.

14.2.4 Example 4 – Multitable SQL* Plus Report Code :

Listing 4. Multitable SQL* Plus Report Code :

```
1:    column OBJECT_TYPE format a20 heading 'Object'
2:    column OBJECT_NAME format a8 heading 'Name'
3:    column COMMENT format a8 heading 'Comments'
4:    break on OBJECT_TYPE skip 1
5:    ttitle 'System Summary Report
```

```
6:  select 'Program' OBJECT_TYPE, program_name OBJECT_NAME,
7:  program_comments COMMENTS
8:  from program_table
9:  UNION
10: select 'Command Language',cl_name, assoc_system
11: from cl_table
12: UNION
13: select 'Files',file_name, 'File Size = ' || file_size || 'Bytes'
14: from file_table
15: /
```

The following code shows the output report from Listing 4.

```
                 System Summary Report
Object                  Name        Comments
------------------------ ----------  ------------------------
Programs                AM1         Algebra Test 1
                        AM2         Algebra Test 2
                        AM3         Algebra Test 3
Command Language        CL1         AM1
                        CL2         AM2
                        CL3         AM3
Files                   AM1.TST     File Size = 1200 Bytes
                        AM2.TST     File Size = 3000 Bytes
                        AM3.TST     File Size = 2200 Bytes
```

Listing 4 creates a SQL* Plus report utilizing different columns from different tables using the SQL UNION command. In this example, there are three different tables (see lines 8, 11, and 14), but there are only three columns of output. The first query contains the column names (see lines 6 and 7). This is because of the way the UNION operator works. The queries after the first query must follow the number of columns and the type of column (text or numeric) based on the column definitions of the first query. The BREAK command (line 4) causes the OBJECT_NAME to print once and creates the blank line between the groupings of records.

I will demonstrate two methods of creating reports that print with specific text in specific positions. Method 1 in Listing 5 utilizes the RPAD SQL function whereas Method 2 in Listing 6 utilizes the COLUMN formatting command. Both examples will create the same output report.

14.2.5 Example 5 & 6 – Fixed Position Formatting SQL* Plus Report Code :

Listing 5. Method 1 Fixed Position Formatting SQL* Plus Report Code :

```
1: define TICKET_ROWID = &1
2: set LINESIZE 80
```

```
set HEADING OFF
set FEEDBACK OFF
spool TICKET.OUT
select RPAD('-------------------------------------------------||null,80),
RPAD('Customer Contact Survey' || null,80),
RPAD('----------------------------------------------–' || null,80),
RPAD('Customer Name:' || CUSTOMER_NAME || 'PHONE#:' ||
PHONE || null,80),
RPAD('Customer Address:' || CUSTOMER_ADDRESS || null,80),
RPAD('   ' || CUSTOMER_CITY || CUSTOMER_STATE ||
CUSTOMER_ZIP || null,80),
RPAD('---------------------------------------------' || null,80),
RPAD(' ' || TO_CHAR(CONTACT_DATE,'mm/dd/yy HH:MI') || '
Caller: ' || CALLER || null,80),
RPAD('---------------------------------------------' || null,80),
RPAD(' Home Phone? ' || HPHONE_YN || 'Best Time to call: '
|| CALL_TIME || null,80),
RPAD(' Has Catalog? ' || CATALOG_YN || 'Desire Future Calls?
' || FUTURE_YN ||null,80),
RPAD('---------------------------------------------' || null,80),
RPAD('PRINTED: ' || TO_CHAR(SYSDATE,'mm/dd/yy HH:MI ||
'BY: ' ||OPERATOR || null,80) from CUSTOMER_TABLE where
ROWID = '&&TICKET_ROWID'
/
set PAGESIZE 1
set NEWPAGE 0
select null from dual;
set PAGESIZE 0
spool OUT
exit
define TICKET_ROWID = &1
set PAGESIZE 55
set LINESIZE 80
set HEADING OFF
set FEEDBACK OFF
column LINE1 JUSTIFY LEFT NEWLINE
column LINE2 JUSTIFY LEFT NEWLINE
column LINE3 JUSTIFY LEFT NEWLINE
column LINE4 JUSTIFY LEFT NEWLINE
column LINE5 JUSTIFY LEFT NEWLINE
column LINE6 JUSTIFY LEFT NEWLINE
column LINE7 JUSTIFY LEFT NEWLINE
column LINE8 JUSTIFY LEFT NEWLINE
column LINE9 JUSTIFY LEFT NEWLINE
column LINE10 JUSTIFY LEFT NEWLINE
column LINE11 JUSTIFY LEFT NEWLINE
column LINE12 JUSTIFY LEFT NEWLINE
column LINE13 JUSTIFY LEFT NEWLINE
column LINE14 JUSTIFY LEFT NEWLINE
break ON ROW SKIP PAGE
SPOOL TICKET
select '------------------------------------------ ' || null LINE1,
'Customer Contact Survey' || null LINE2,
' Customer Name: ' || CUSTOMER_NAME || ' PHONE#: ' ||
PHONE || null LINE4,
' Customer Address: ' || CUSTOMER_ADDRESS || null LINE5,
'|| CUSTOMER_CITY || CUSTOMER_STATE ||
CUSTOMER_ZIP || null LINE6,
'-----------------------------------------------–' || null LINE7,
' ' || TO_CHAR(CONTACT_DATE,'mm/dd/yy HH:MI || ' Caller:
' || CALLER || null
LINE8,
'-----------------------------------------------–' || null LINE9,
'Home Phone? ' || HPHONE_YN || 'Best Time to call: ' || CALL_TIME
|| null
LINE10,
' 'Has Catalog? ' || CATALOG_YN || 'Desire Future Calls? ' ||
FUTURE_YN || null
LINE11,
'------------------------------------------------ ' || null LINE12,
'PRINTED: ' || TO_CHAR(SYSDATE,'mm/dd/yy HH:MI || 'BY: '
|| OPERATOR || null
LINE13,
'-----------------------------------------------–' || null LINE14
from CUSTOMER_TABLE
where ROWID = '&&TICKET_ROWID'
/
spool OUT
exit
```

Listings 5 and 6 both produce the same output report.

❑ **Check Your Progress – 1 :**

1. A parent–child relationship between two tables can be created only, when there is a ________ in one table and FOREIGN KEY in another table.

   a. PRIMARY KEY b. SECONDARY KEY

14.3 Let Us Sum Up :

In this unit you learned the history and functional uses of SQL* Plus and saw an in–depth list of SQL* Plus commands with examples. You used these commands in a variety of ways to produce report and program output examples. Some of the features discussed in this chapter are not directly referenced in Oracle documentation.

Hopefully, you can utilize the skills and refer to the examples provided in this chapter in your application, design, and development of Oracle–based products.

14.4 Answers for Check Your Progress :

❑ **Check Your Progress 1 :**

1 : a

14.5 Glossary :

Constant : It is a value used in a PL/SQL Block that remains unchanged throughout the program.

14.6 Assignment :

The following relations keep track of airline flight information :

- Flights (flno: integer, from: string, to: string, distance: integer, departs: time, arrives: time, price: integer)
- Aircraft(aid: integer, aname: string, cruisingrange: integer)
- Certified(eid: integer, aid: integer)
- Employees(eid: integer, ename: string, salary: integer)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly. Write each of the following queries in SQL.

1 Find the names of aircraft such that all pilots certified to operate them earn more than $80,000.

2 For each pilot who is certified for more than three aircraft, find the eid and the maximum cruising range of the aircraft for which she or he is certified.

3 Find the names of pilots whose salary is less than the price of the cheapest route from Los Angeles to Honolulu.

4 For all aircraft with cruising range over 1000 miles, find the name of the

5 aircraft and the average salary of all pilots certified for this aircraft.

6 Find the names of pilots certified for some Boeing aircraft.

7 Find the aids of all aircraft that can be used on routes from Los Angeles to Chicago.

8 Identify the routes that can be piloted by every pilot who makes more than $100,000.

9 Print the enames of pilots who can operate planes with cruisingrange greater than 3000 miles but are not certified on any Boeing aircraft.

10 A customer wants to travel from Madison to New York with no more than two changes of flight. List the choice of departure times from Madison if the customer wants to arrive in New York by 6 p.m.

11 Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots).

12 Print the name and salary of every nonpilot whose salary is more than the average salary for pilots.

13 Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles.

14 Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles, but on at least two such aircrafts.

14.7 Activities :

Consider the following relational schema. An employee can work in more than one department; the pct_time filed of the Works relation shows the percentage of time that a given employee works in a given department.

- Emp(eid: integer, ename: string, age: integer, salary: real)
- Works(eid: integer, did: integer, pct_time: integer)
- Dept(did: integer, dname: string, budget: real, managerid: integer)
- Write the following queries in SQL :

1 Print the names and ages of each employee who works in both the Hardware department and the Software department.

2 For each department with more than 20 full–time–equivalent employees (i.e., where the part–time and full–time employees add up to at least that many fulltime employees), print them together with the number of employees that work in that department.

3 Print the name of each employee whose salary exceeds the budget of all of the departments that he or she works in.

4 Find the manager ids of managers who manage only departments with budgets greater than $1 million.

   a. Find the enames of managers who manage the departments with the largest budgets.

   b. If a manager manages more than one department, he or she controls the sum of all the budgets for those departments. Find the manager ids of managers who control more than $5 million.

   c. Find the manager ids of managers who control the largest amounts.

Find the enames of managers who manage only departments with budgets larger than $1 million, but at least one department with budget less than $5 million.

14.8 Case Study :

Case 1 demonstrates the following :

A simple query

A simple query retrieving full date information

The first query retrieves all the data from GTW_DEPT and confirms that the gateway is working correctly. The second query retrieves all the data from GTW_EMP including the time portion of the hire date because the default date format was set to DD–MON–YY HH:MM:SS for the session by an ALTER SESSION command.

To run Case 1, log on to SQL* Plus as SCOTT/TIGER and enter the following:

```
SQL> START CASE1
```

Case 1 executes two SQL statements. The first statement is as follows:

SELECT * FROM GTW_DEPT@GTWLINK; which results in the following :

```
DEPTNO      DNAME           LOC
----------  --------------  -------------
10          ACCOUNTING      NEW YORK
20          RESEARCH        DALLAS
30          SALES           CHICAGO
40          OPERATIONS      BOSTON
```

The following command and SQL statement change the date format to DD–Mon–YY HH24:MM:SS in Oracle and retrieve the employee name and hire date from the Rd? database:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD–Mon–YY HH24:MM:SS';
SELECT ENAME, HIREDATE FROM GTW_EMP@GTWLINK;
```

14.9 Further Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth –Tata McGraw – Hill Publication.
3. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
4. Database Management System by Raghu Ramkrishnan – Tata McGraw – Hill Publication.
5. SQL, PL/SQL: The Programming Language Oracle – Ivan Bayross – BPB Publication.

Unit 15 PRACTICAL SOLVING OF SQL

UNIT STRUCTURE

15.0 Learning Objectives :

After learning this unit, you will be able to understand :

- Various examples using PL/SQL
- Creating batch files using MS–DOS and Master–Detail form creation which can be useful in database applications

15.1 Introduction :

The purpose of any exercise is to steadily develop skills and to acquire the automatic algorithms for fulfilling certain operations. As for the SQL language, practical exercises are intended to enable the database programmer to quickly devise SQL queries in order to solve practically any problem, by having already studied similar problems in exercises. This unit provides such a collection of practical solved SQL exercises.

15.2 Practical Solving Using SQL :

The following example formats the results of a SQL query. It defines a report title and formats, assigns column headings, and applies some control breaks for intermediate and report totalling.

15.2.1 Example 1 – Customer Contact Survey :

```
----------------------------------------------------------------------
                    Customer Contact Survey
----------------------------------------------------------------------
Customer Name : John Smith PHONE#:
515 123–4567 Customer Address: 123
Oak Street Anytown VA 12345
----------------------------------------------------------------------
31–Aug–95 10:05 Caller: ABC
----------------------------------------------------------------------
Home Phone?  Y  Best Time to call: 8pm
Has Catalog?   Y  Desire Future Calls? N
----------------------------------------------------------------------
               PRINTED: 31–Aug–95 12:45 BY: ABC
```

These examples utilize the concatenation feature of SQL (||) to blend text between database fields. Each column in the SQL statement represents an individual line in the report. Both examples have the standard column headings feature turned off. Both examples have a one–to–one relationship between a SQL column and a line of output. The methods differ in how the columns are formatted to create the individual lines.

The main difference in these two methods is the approach used in the individual line setup. Method 1 uses the SQL command RPAD in combination with LINESIZE to create an output line. The RPAD is used to fill the line with blanks to position 80, and with LINESIZE set at 80 will cause the formatted line to appear on a line by itself. Method 2 uses the column command with the option NEWLINE specified in conjunction with a field alias name. The column command with the NEWLINE option will make the formatted line appear on a line by itself.

15.2.2 Example 2 – SQL Creating SQL :

The classic example of using SQL* Plus formatting to create other SQL statements (hence the term "SQL creating SQL") is cleaning up a table after an employee leaves a company. The Oracle data dictionary view TAB is used in this example. You can easily enter at the SQL* Plus prompt (shown here as SQL>) the steps in Listing 8 or adapt them to a SQL* Plus command file using features you already learned.

Listing 8. Dropping all tables owned by a particular user.

```
SQL>set headings off
SQL>set pagesize 0
 SQL>set termout off
SQL>spool drop_tbl.sql
SQL>select 'DROP TABLE ' || tname || ';' from tab;
SQL>spool off
SQL>set termout on
SQL>start drop_tbl
```

This scenario assumes that the login ID and the owner of the table objects to be dropped are both the same. The first three commands are used to set up the SQL* Plus environment. The spool file drop_tbl.sql will capture the concatenated text and table names (tname) from the SQL query. The spool off command closes the file and the start command executes the drop table commands now inside the drop_tbl.sql file.

15.2.3 Example 3 – SQL Creating Database Triggers :

Listing 3 is an extension of Listing 2 as another example of creating useful database–driven programs. This example will add four auditing fields to the end of each table owned by the user ID that runs this particular SQL* Plus command file. This script will also create a database trigger that will automatically maintain these four added fields. I utilized the fixed position formatting discussed in Listing

Listing 3. SQL Creating Database Triggers :

```
set ECHO OFF
set TERMOUT OFF
FEEDBACK OFF
set VERIFY OFF
set PAGESIZE 0
set LINESIZE 80
set HEADING OFF
spool cre_dbtrg.sql
select RPAD('select ' alter table ' || TNAME || null,80),
RPAD('    add (inserted_by    varchar2(10), ' || null,80),
RPAD('    inserted_date    date    , ' || null,80),
RPAD('    updated_by    varchar2(10), ' || null,80),
RPAD('    updated_date    date    ); ' || null,80)
from TAB;
Select RPAD(' create trigger trg_' || TNAME || null,80),
RPAD(' before insert or update ' || null,80),
RPAD('on ' || TNAME || null,80),
RPAD(' for each row ' || null,80),
RPAD(' begin ' || null,80),
RPAD(' if :old.inserted_by is null then    ' || null,80),
RPAD('    :new.inserted_by    := USER;    ' || null,80),
RPAD('    :new.inserted_date    := SYSDATE;    ' || null,80),
RPAD('    :new.updated_by    := null;    ' || null,80),
RPAD('    :new.updated_date    := null;    ' || null,80),
RPAD('    else ' || null,80),
26:   RPAD('    :new.inserted_by     := :old.inserted_by;     ' || null,80),
27:   RPAD('   :new.inserted_date    := :old.inserted_date;    ' || null,80),
28:   RPAD('    :new.updated_by      := USER;     ' || null,80),
29:   RPAD('    :new.updated_date      := SYSDATE;     ' || null,80),
30:   RPAD('    end if; ' || null,80),
31:   RPAD('    end; ' || null,80),
32:   RPAD( '/' || null,80)
33:   from TAB;
      spool off
34:   set FEEDBACK ON
35:   set TERMOUT ON
36:   set VERIFY ON
37:   set ECHO ON
38:   spool dbtrg.log
39:   start dbtrg.sql
40:   spool off
41:   exit
```

Lines 1 through 7 set up the SQL* Plus environment so that no extra messages appear in the cre_dbtrg.sql file (see line 8). Lines 9 through 14 create the SQL alter table statement that will add the audit fields to each table, and lines 15 through 33 create the SQL create trigger statement that will add the database triggers necessary to maintain these audit fields. Lines 35 through 38 reset the SQL* Plus environment so that all SQL commands and messages display. Line 40 then runs the SQL* Plus command file cre_dbtrg.sql that was just created.

Important Note : In Listing 9, line 39 opens the file DBTRG.LOG. This file will contain the output (an audit trail) when the DBTRG.SQL statement is executed with the START command on Line 40. I like to create SQL audit trails for various DBA commands, particularly ones such as this example where the process is rather automated. The audit trails enable me to review the additions and any errors that might have occurred by simply editing the log file.

15.2.4 Example 4 – SQL Creating Command Language Scripts :

The example in Listing 4 applies the SQL creating SQL discussed in Listing 3 to create a DOS BAT file.

Listing 4. SQL Creating Command Language Scripts :

```
1:    column HOME_DIR new_value HDIR noprint
2:    column PROGRAM_DIR new_value PDIR noprint
3:    column PROGRAM_SUFFIX new_value PSUF noprint
4:    select HOME_DIR,PROGRAM_DIR,PROGRAM_SUFFIX
5:    from APPLICATION_DEFAULTS
6:    /
```

```
7:    spool LIST614.BAT
8:
9:    select 'CD &PDIR'
10:   from dual
11:   /
12:   select 'DIR *.&PSUF'
12:   from dual
13:   /
14:   select 'CD &HDIR'
15:   from dual
16:   /
17:   spool off
18:   exit
```

The following code is the output created by Listing 6.10.

```
CD \COBOL\PROGRAMS

DIR *.COB

CD \
```

Listing 4 is a simple example of creating an MS–DOS batch file with SQL* Plus formatting commands. The important concept of this example comes in lines 1 through 3. These lines contain three column commands that contain the NEW_VALUE clause. The importance of this concept is that these variables can be loaded from the Oracle database and their values referenced again in other SQL queries. Lines 4 and 5 populate these variables as named in the column statement. Note that when the variables are referenced in other SQL queries (lines 8, 11, and 14), the reference is to the NEW_VALUE variable name.

Important Note : Use the column command with the NEW_VALUE option to load variables from Oracle tables to use in other SQL queries.

15.2.5 Example 5 – MS–DOS Batch Command File :

Listing 5. MS–DOS Batch Command File :

```
1:    SQLPLUS –S HOTKA/DAN @LIST6_16.SQL
2:    CALL LIST6_16.BAT
3:    SED –F LIST6_19.SED LIST6_15A.DAT > LIST6_15B.DAT
4:    SQLLOAD USERID=HOTKA/DAN CONTROL=LIST6_16.CTL
```

Listing 5 is the actual MS–DOS bat command file that runs the four computer tasks to accomplish our goal. The SQLPLUS command on line 1 connects to the database and runs the SQL* Plus command file LIST6_16.SQL LIST6_16.SQL creates two files, LIST6_16.BAT and LIST6_16.CTL Line 2 executes the newly created LIST6_16.BAT file. This command creates the file, LIST6_15A.DAT, that is an MS–DOS DIR (directory) list of directory 'C:\COBOL'. Line 3 is a stream editor (SED) that deletes the first few lines and the last few lines (as directed by LIST6_19.SED; see Listing 15) of file LIST6_15A.DAT, creating LIST6_15B.DAT. This file is the MS–DOS DIR output without the heading and trailing text information. Line 4 then runs Oracle's SQL* Loader program, using the LIST6_16.CTL SQL* Loader control file created by line 1 and reading the datafile LIST6_15B.DAT file created by line 3.

LIST6_16.SQL referenced in Line 1 of Listing 11 and will create the LIST6_16.BAT file referenced in Line 2.

15.2.6 Example 6 – SQL* Plus Command File LIST6_16.SQL :

Listing 6. SQL* Plus Command File LIST6_16.SQL :

```
1:   set PAGESIZE 0
2:   column HOME_DIR new_value HDIR noprint
3:   column PROGRAM_DIR new_value PDIR noprint
4:   column PROGRAM_SUFFIX new_value PSUF noprint
5:   select HOME_DIR,PROGRAM_DIR,PROGRAM_SUFFIX
6:   from APPLICATION_DEFAULTS
7:   /
8:   spool LIST6_16.BAT
9:   select 'DIR &PDIR\*.&PSUF >&HDIR\LIST6_15A.DAT'
10:  from dual
11:  /
12:  spool off
13:  spool LIST6_16.ctl
14:  select 'load data'
15:  from dual
16:  /
17:  select 'infile '|| "" || 'LIST6_15B.DAT' || ""
18:  from dual
19:  /
20:  select 'append'
21:  from dual
22:  /
23:  select 'into table APPLICATION_PROGRAMS'
24:  from dual
25:  /
26:  select '(PROGRAM_NAME position(1:8) char,'
27:  from dual
28:  /
29:  select 'PROGRAM_SUFFIX constant ' || "" || '&PSUF' || "" || ','
30:  from dual
31:  /
select ÔPROGRAM_SIZE position(15:22) integer external,'
from dual
/
select 'PROGRAM_PATH constant ' || "" || '&PDIR' || "" || ','
from dual
/
select 'ASSIGNED_ANALYST constant ' || "" || '&USER' || "" || ')'
from dual
/
spool off
exit
```

15.2.7 Master/Detail or Parent/Child SQL :

A parent–child relationship between two tables can be created only, when there is a PRIMARY KEY in one table and FOREIGN KEY in another table.

Master–detail forms display a master row and multiple detail rows within a single HTML page. With this form, users can query, insert, update, and delete values from two tables or views. The tables or views are joined together in the wizard by a SQL JOIN condition.

Values in the master row determine which detail rows are displayed for updating. For example, the master row could display column values from the SCOTT.DEPT table. This table joins to SCOTT.EMP by a column in each table called DEPTNO. Selecting a department ID in the master row displays in detail rows all Employees having the same Department ID.

The end user of the master–detail form can update or delete values in the master row. The wizard provides an option to allow cascading deletes, so that deleting a master row also deletes detail rows. The same master–detail form can be used to insert new detail rows, or update and delete existing detail rows.

15.3 Let Us Sum Up :

In this unit you learned the history and functional uses of SQL* Plus and saw an in–depth list of SQL* Plus examples. You used these commands in a variety of ways to produce report and program output examples. Some of the features discussed in this chapter are not directly referenced in Oracle documentation.

Hopefully, you can utilize the skills and refer to the examples provided in this chapter in your application, design, and development of database applications.

15.4 Glossary :

A parent–child relationship between two tables can be created only, when there is a PRIMARY KEY in one table and FOREIGN KEY in another table.

15.5 Assignment :

Create a database application for ecommerce website

15.6 Activities :

Create a master detail form for Invoice Application

15.7 Case Study :

Create a list of reports that can be useful for an ecommerce application

15.8 Further Readings :

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth –Tata McGraw – Hill Publication.
3. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
4. Database Management System by Raghu Ramkrishnan – Tata McGraw – Hill Publication.
5. SQL, PL/SQL : The Programming Language Oracle – Ivan Bayross – BPB Publication.

BLOCK SUMMARY :

This chapter is about PL/SQL programming. The following PL/SQL subject topics were discussed: overview of PL/SQL, modular coding principles, building PL/SQL locks, communicating with Oracle, and process control. Conditional statements and loops, cursors, and error handling help you flow. PL/SQL is the most up-to-date approach for writing and managing stored procedures. Using data from Oracle the declaration is the first subblock in PL/SQL code. Furthermore, PL/SQL is used in four different programming constructs. Procedures are the several categories. They both include a set of instructions that will be executed by PL/SQL. Nonetheless, the primary distinction is that a function will always return a single value.

BLOCK ASSIGNMENT :

❖ **Short Questions :**

1. Package Creation
2. Variables
3. Constants
4. Data types and Arrays
5. Control Statements

❖ **Long Questions :**

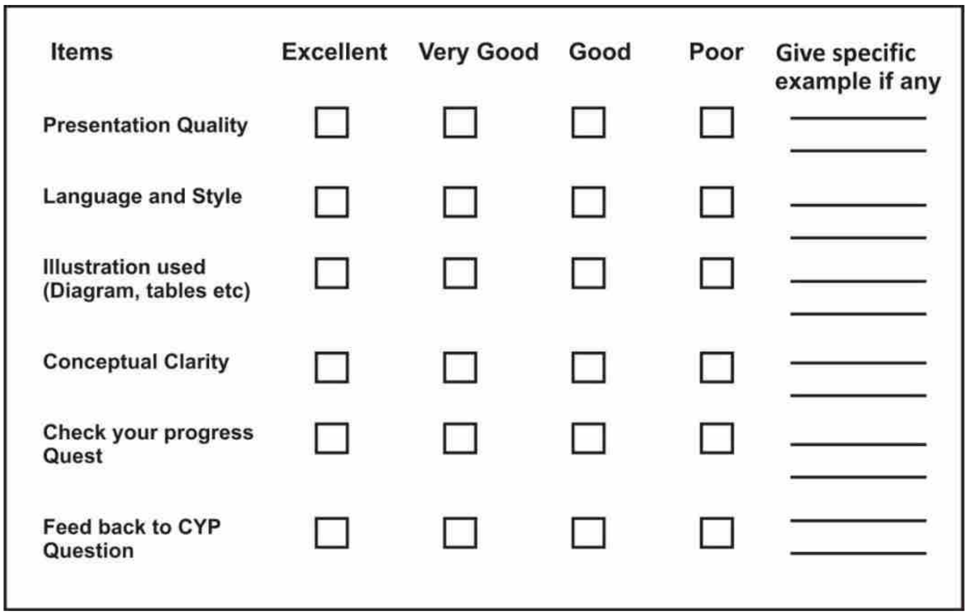
1. What do you understand by control statements ?
2. What do you understand by package subprograms ?
3. Discuss the naming conventions.

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	12	13	14	15
No. of Hrs.				

2. Please give your reactions to the following items based on your reading of the block :



Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	________
Language and Style	☐	☐	☐	☐	________
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	________
Conceptual Clarity	☐	☐	☐	☐	________
Check your progress Quest	☐	☐	☐	☐	________
Feed back to CYP Question	☐	☐	☐	☐	________

3. Any other Comments

....

....

....

....

....

....

....

....