



திருவள்ளூர் பல்கலைக்கழகம்
THIRUVALLUVAR UNIVERSITY

(State University Accredited with "B" Grade by NAAC)
Serkkadu, Vellore - 632 115, Tamil Nadu, India.

E-NOTES / CS & BCA

THIRUVALLUVAR UNIVERSITY



E-NOTES

BCS 41/ BCA 41

DATABASE MANAGEMENT SYSTEMS

(4th Semester B.Sc. CS / B.C.A.)

STEERED BY

Dr. S. Thamarai Selvi, M.E., Ph.D.,
Vice Chancellor, Thiruvalluvar University, Serkkadu, Vellore

PREPARED BY

- 1. Dr. A.PRIYA, M.Sc., M.Phil., Ph.D.,**
Assistant Professor & Head, PG Department of Computer Science,
Thiruvalluvar University College of Arts and Science, Tirupattur.
- 2. Mrs. A.POORNIMA, M.Sc., M. Phil., (SET), M.B.A.**
Assistant Professor, Department of Computer Science,
Marudhar Kesari Jain College for Women, Vaniyambadi.
- 3. Mrs. G. SASIREKHA M.Sc., M. Phil., B.Ed.,**
Assistant Professor, Department of Computer Science,
Marudhar Kesari Jain College for Women, Vaniyambadi.
- 4. Mr. A.ANTONY PAUL RAJ, M.Sc., M.Phil., B.Ed., (Ph.D.)**
Assistant Professor, Department of Computer Science,
BWDA Arts and Science College, Villupuram.
- 5. Dr. S.LAVANYA, M.Sc., M.Phil., Ph.D.,**
Assistant Professor & Head, Department of Computer Applications,
Auxilium College (Autonomous), Vellore.
- 6. Ms. D.KAVITHA, M.Sc., M.Phil.,**
Assistant Professor & Head-In-Charge,
Department of Computer Applications,
M.M.E.S. Women's Arts and Science College, Melvisharam



ACKNOWLEDGEMENT



*Writing an e-content is harder than we thought and more rewarding than we could have ever imagined. None of this would have been possible without our honourable Vice Chancellor – Professor **Dr. S. Thamarai Selvi**. Her dynamism, vision, sincerity, guidance and motivation have deeply inspired us. She worked her magic on every page in this e-content, reorganised content wherever necessary, bringing value added words in every line. She has taught us the way to present the content to the students in a best possible way to have clear and crisp understanding. She was able to tune into the task and identify the ones that diverted our team focus and energy keeping us from getting our things done. It was a great privilege and honour to work under her guidance. We are extremely grateful for what she has offered us.*



Dr. A. PRIYA



Mrs. A. POORNIMA



Mrs. G. SASIREKHA



Mr. A. ANTONY PAUL RAJ



Dr. S. LAVANYA



Ms. D. KAVITHA



SYLLABUS

Objective: To incorporate a strong knowledge on databases to students

UNIT – I: Database Basics:

Introduction: Flat File – Database System – Database – Actionable for DBA. The Entity – Relationship Model: Introduction – The Entity Relationship Model. Data Models: Introduction – Relational Approach – The Hierarchical Approach – The Network Approach.

UNIT – II: Relational Algebra:

Structure of Relational Databases – Fundamental Relational Algebra Operations – Additional Relational Algebra Operations - Extended Relational Algebra Operations - Null Values - Modification of the Database - The Tuple Relational Calculus – The Domain Relational Calculus.

UNIT – III: Normalization:

Normalization: Introduction - Normalization – Definition of Functional Dependence (FD) – Normal Forms: 1NF, 2NF, 3NF and BCNF.

UNIT – IV: Structured Query Language:

Structured Query Language: Features of SQL – Select SQL Operations – Grouping the Output of the Query – Querying from Multiple Tables – Retrieval Using Set operators – Nested Queries. T-SQL – Triggers and Dynamic Execution: Transact-SQL.

UNIT – V: Procedural Language:

Procedural Language- SQL: PL/SQL Block Structure – PL/SQL Tables. Cursor Management and Advanced PL/SQL: Opening and Closing a Cursor – Processing Explicit Cursor – Implicit Cursor – Exception Handlers – Sub Programs in PL/SQL



– Functions – Precaution While Using PL/SQL Functions – Stored Procedure – Object Oriented Technology.

Text Book:

1. Rajesh Narang, "Database Management Systems", PHI Learning Private Limited, New Delhi, Sixth Printing, 2010.

Reference Books:

1. S.K. Singh, "Database Systems – Concepts, Design and Applications", Dorling Kindersley (India) Pvt. Ltd., Second Impression, 2008
2. Abraham Silberchatz, Henry F Korth, S.Sudarshan, "Database System Concepts", Tata McGraw-Hill - 5th Edition - 2006.
3. Raghu Ramakrishnan and Johannes Gehrke, "Database Management Systems", Tata McGraw-Hill Publishing Company, 2003.
4. Ramez Elmasri and Shamkant B. Navathe, "Fundamental Database Systems", Pearson Education, Third Edition, 2003.
5. Hector Garcia–Molina, Jeffrey D.Ullmanand Jennifer Widom, "Database System Implementation", Pearson Education - 2000.
6. S.K .Singh, "Database systems: Concepts, Design &Applications", Pearson Education.
7. Gerald V. Post, "DBMS – Designing and Business Applications", Tata McGraw Hill Publication.
8. Micheal Abbey And Micheal.J.Corey, "Oracle A Beginners guide", Tata McGraw-Hill Publication.



INDEX

CHAPTER NO.	CONTENT	PAGE NO
	Chapter 1: Database Basics	12
1.1.	Introduction	12
1.2.	Flat File	12
1.3.	Database System	13
1.3.1.	Data	13
1.3.2.	Software	14
1.3.3.	Hardware	14
1.3.4.	Users	14
1.4.	Database	16
1.4.1.	Types of Databases	17
1.5.	Database Languages	19
1.5.1.	Data Definition Language (DDL)	19
1.5.2.	Data Manipulation Language (DML)	20
1.6.	Actionable For DBA	21
	Review Questions	23
	Objective Type Questions	24
	Chapter – 2: The Entity – Relationship Model	32
2.1.	Introduction	32
2.2.	The Entity Relationship Model	33
2.2.1.	Components of an E-R Diagram	34



2.2.2.	Various Terms Related To Relationships	36
2.3.	Keys	38
2.3.1.	Types of Key	38
	Review Questions	40
	Objective Type Questions	40
	Chapter – 3 : Data Models	43
3.1.	Introduction	43
3.2.	Types Of Data Models	43
	Review Questions	48
	Objective Type Questions	50
	Chapter 4: Relational Algebra	52
4.1.	Introduction	52
4.2.	Structure of Relational Databases	53
4.2.1.	Select Operation	53
4.2.2.	Project Operation	54
4.2.3.	Union Operation	56
4.2.4.	Set Intersection	58
4.2.5.	Set Difference	58
4.2.6.	Cartesian Product	59
4.2.7.	Rename Operation	60
4.3.	Fundamental Relational Algebra Operations	60
4.3.1.	Join Operations	60
4.3.2.	Types Of Join Operations	62



4.4.	Additional Relational Algebra Operations	67
4.4.1.	Set-Intersection	67
4.4.2.	Theta Join	67
4.5.	Extended Relational Algebra Operations	68
4.5.1.	Generalized Projection	68
4.6.	Null Values	68
4.7.	Modification Of The Database	69
4.7.1.	Deletion	69
4.7.2.	Insertion	69
4.7.3.	Updation	69
4.8.	Tuple Relational Calculus	70
4.9.	Domain Relational Calculus	72
	Review Questions	74
	Objective Type Questions	82
	Chapter 5: Normalization	85
5.1.	Introduction	85
5.2.	What Is Normalization?	85
5.3.	Functional Dependency	88
5.3.1.	Full Functional Dependency	89
5.3.2.	Partial Dependency	91
5.3.3.	Transitive Dependency	91
5.3.4.	Multi-Valued Dependency	92
5.3.5.	Join Dependency	92



5.4.	Normal Forms	92
5.4.1.	First Normal Form (1 NF)	92
5.4.2.	Second Normal Form (2 NF)	94
5.4.3.	Third Normal Form (3 NF)	96
5.4.4.	Boyce- Code Normal Form (BCNF)	98
5.4.5.	Forth Normal Form (4NF)	100
5.4.6.	Join Dependencies And Fifth Normal Form (5NF)	102
5.4.7.	Domain Key Normal Form (DKNF)	105
	Review Questions	106
	Objective Type Questions	107
	Chapter 6: Structured Query Language	109
6.1.	Introduction	109
6.2.	Structured Query Language	109
6.2.1.	Need for SQL	110
6.2.2.	Types of SQL Statements	110
6.3.	Features of SQL	111
6.4.	Select SQL Operations	112
6.5.	Grouping The Output Of The Query	114
6.5.1.	SQL Group Functions	114
6.5.2.	SQL Group By Clause	115
6.5.3.	SQL Having Clause	116
6.6.	Querying From Multiple Tables	117
6.6.1.	SQL Subquery	117



6.6.2.	Correlated Subquery	121
6.7.	Retrieval Using Set Operators	122
6.7.1.	Union Operation	122
6.7.2.	Union All	123
6.7.3.	Intersect	124
6.7.4.	Minus	125
	Review Questions	126
	Objective Type Questions	135
	Chapter 7: T-SQL – Triggers And Dynamic Execution	138
7.1.	Introduction	138
7.2.	T-SQL (Transact-SQL)	138
7.2.1.	T-SQL Functions	139
7.2.2.	Transact-SQL	141
	Review Questions	141
	Chapter 8: Procedural Language	142
8.1.	Introduction	142
8.2.	PL/ SQL Block Structure	143
8.2.1	PL/SQL Block Syntax	146
8.2.2.	Types of PL/SQL Block	146
8.2.3.	Fundamentals Of PL/SQL	149
8.3.	PL/SQL - Data Types	151
8.3.1.	Scalar Data Types	152
8.3.2.	PL/SQL Composite Data Types	154



8.3.3.	Reference Data Types	164
8.3.4.	Large Object Datatypes (LOB Types)	166
8.3.5.	Unknown Column Types	167
8.3.6.	User-Defined Subtypes	167
8.4.	PL/SQL - Variables	168
8.5.	PL/SQL - Operators	172
8.5.1.	Arithmetic Operators	172
8.5.2.	Relational Operators	173
8.5.3.	Comparison Operators	173
8.5.4.	Logical Operators	174
8.5.5.	PL/SQL Operator Precedence	175
8.6.	PL/SQL - Conditions	176
8.7.	PL/SQL - Loops	178
8.8.	PL/SQL – Strings	181
8.8.1.	Declaring String Variables	182
8.8.2.	PL/SQL String Functions And Operators	183
8.9.	PL/SQL – Arrays	184
8.10.	PL/SQL Tables And Records	190
8.10.1.	Tables	191
8.10.2.	Records	194
8.11.	PL/SQL Cursor	199
8.11.1.	PL/SQL Implicit Cursors	199
8.11.2.	PL/SQL Explicit Cursors	202



8.12.	PL/SQL Exception Handling	206
8.12.1.	PL/SQL System-Defined Exceptions	206
8.12.2.	PL/SQL User-Defined Exceptions	212
8.13.	PL/SQL Sub Programs	213
8.13.1.	PL/SQL Functions	213
8.13.2.	PL/SQL Procedure	217
	Review Questions	225
	Objective Type Questions	226
	Chapter 9: Object Oriented Technology	230
9.1.	Introduction	230
9.2.	Object Oriented Features	232
9.3.	Components	233
9.3.1.	Messages	233
9.3.2.	Methods	233
9.3.3.	Variables	233
9.4.	Object Classes	234
	Review Questions	235
	Objective Type Questions	235
	Appendix A: ODBC CONNECTIVITY	236
	Appendix B: JDBC CONNECTIVITY	251
	Appendix C: SAMPLE PROGRAMS / APPLICATIONS	284
1.	Table Creation And Simple Queries	284
2.	Queries Using Aggregate Function And Set Operations	292



3.	Table Creation With Various Joins	297
4.	Nested Sub Queries And Correlated Sub Queries	299
5.	View Creation And Manipulation	300
6.	PL/SQL Program For Cursor	302
7.	PL/SQL Program For Packages	304
8.	PL/SQL Program For Triggers And Its Type	305
		307
9.	PL/SQL Program For Procedures And Functions	



UNIT – I

CHAPTER 1: DATABASE BASICS

1.1. INTRODUCTION

DATA: Facts and statistics collected together for reference or analysis.

For example: 5, "Ravi", 234.32

DATABASE: A **database** is a collection of information that is organized so that it can be easily accessed, managed and updated. Computer **databases** typically contain aggregations of data records or files, containing information about any transactions or interactions.

For example:

Register Number	Student Name	Department	Age
35518U18001	Ashwin	CS	19
35518U09023	Murugan	BCA	18
35518U18032	Pavithra	CS	20
35518U09040	Shreenath	BCA	18
35518U18050	Varshi	CS	17

DBMS: A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data. Software such as DBASE IV or V, Microsoft ACCESS, or EXCEL

GOAL OF DBMS: The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

1.2. FLAT FILE

A flat file database is a type of database that stores data in a single table. This is unlike a relational database, which makes use of multiple tables and relations. Flat file databases are generally in plain-text form, where each line holds only one record. The fields in the record are separated using delimiters such as tabs and commas.



Flat file database tables can be set in various application types, including HTML documents, simple word processors or worksheets in spreadsheet applications. The tables within a flat file database can be sorted based on column values. These tables serve as a solution for simple database tasks.

In spite of the limitations associated with flat files, flat file databases are used internally by various computer applications to store data related to configuration. Most of the applications permit users to store and retrieve information from flat files based on a predefined set of fields. Flat files include data types common to other databases. A few features of the data in flat file databases include:

- Database Management System: The text data represent an intermediate style of data before being loaded into the database.
- Separated Columns: Flat file databases are based on fixed-width data formatting. Columns are separated using delimiter characters.
- Data Types: Columns in the database tables are restricted to a particular data type and are not indicated, unless the data is passed on to a relational database.
- Relational Algebra: Records in flat file database tables meet tuple definitions under relational algebra.

1.3. DATABASE SYSTEM

A Database System involves four components,

1. Data
2. Software
3. Hardware and,
4. Users

1.3.1. DATA

Data are sub-divided into one or more databases. Each database is a repository or storage of the data. The database is integrated and shared, which means that the database is a unification of several different data files and redundancy among the files is eliminated to the maximum extent.



The term "shared" means the individual data items to database can be shared among several different users. This sharing is possible because the database is integrated. A data item is not just shared by users sequentially but also concurrently. i.e. at the same time. A database system supporting this form of sharing is called multiuser system.

1.3.2. SOFTWARE

Database Management System (DBMS) lies between the physical database and users of the system. Access requests coming from users are handled by DBMS. The database users from hardware-level details and supports user operation by retrieving data.

1.3.3. HARDWARE

It consists of secondary storage volumes – disks, on which the database resides.

1.3.4. USERS

This differentiation is made according to the interaction of users to the database. Database system is made to store information and provide an environment for retrieving information.

There are three types of users:

- (i) An Application Programmers
- (ii) End – User and
- (iii) Database Administrator.

I. APPLICATION PROGRAMMERS

As its name shows, application programmers are the one who writes application programs that uses the database. These application programs are written in programming languages like COBOL or PL (Programming Language 1), Java and fourth generation language. These programs meet the user requirement and made according to user requirements. Retrieving information, creating new information and changing existing information is done by these application programs.



They interact with DBMS through DML (Data manipulation language) calls. And all these functions are performed by generating a request to the DBMS. If application programmers are not there then there will be no creativity in the whole team of Database.

II. END USERS

End users are those who access the database from the terminal end. They use the developed applications and they don't have any knowledge about the design and working of database. These are the second class of users and their main motto is just to get their task done.

There are basically two types of end users that are discussed below.

- a. Casual User
- b. Naive User
 - **Casual User:** These users have great knowledge of query language. Casual users access data by entering different queries from the terminal end. They do not write programs but they can interact with the system by writing queries.
 - **Naïve:** Any user who does not have any knowledge about database can be in this category. Their task is to just use the developed application and get the desired results.

III. DBA (DATABASE ADMINISTRATOR)

DBA can be a single person or it can be a group of person. Database Administrator is responsible for everything that is related to database. He makes the policies, strategies and provides technical supports.

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator (DBA). The functions of a DBA include:

a. SCHEMA DEFINITION

The DBA creates the original database schema by executing a set of data definition statements in the DDL. Storage structure and access-method definition.



b. SCHEMA AND PHYSICAL-ORGANIZATION MODIFICATION.

The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

c. GRANTING OF AUTHORIZATION FOR DATA ACCESS.

By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

d. ROUTINE MAINTENANCE.

Examples of the database administrator's routine maintenance activities are: Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding. Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required. Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

1.4. DATABASE

A database is a data structure that stores organized information. Most databases contain multiple tables, which may each include several different fields. For example, a company database may include tables for products, employees, and financial records. Each of these tables would have different fields that are relevant to the information stored in the table.

A database is a collection of information that is organized so that it can be easily accessed, managed and updated. Computer databases typically contain aggregations of data records or files, containing information about sales transactions or interactions with specific customers.

In a relational database, digital information about a specific customer is organized into rows, columns and tables which are indexed to make it easier to find relevant information through SQL or NoSQL queries.



Typically, the database manager provides users with the ability to control read/write access, specify report generation and analyze usage. Some databases offer ACID (atomicity, consistency, isolation and durability) compliance to guarantee that data is consistent and that transactions are complete.

1.4.1. TYPES OF DATABASES

Databases have evolved since their inception in the 1960s, beginning with hierarchical and network databases, through the 1980s with object-oriented databases, and today with SQL and NoSQL databases and cloud databases.

I. RELATIONAL DATABASE

A relational database, invented by E.F. Codd at IBM in 1970, is a tabular database in which data is defined so that it can be reorganized and accessed in a number of different ways.

Relational databases are made up of a set of tables with data that fits into a predefined category. Each table has at least one data category in a column, and each row has a certain data instance for the categories which are defined in the columns.

The Structured Query Language (SQL) is the standard user and application program interface for a relational database. Relational databases are easy to extend, and a new data category can be added after the original database creation without requiring that you modify all the existing applications.

II. DISTRIBUTED DATABASE

A distributed database is a database in which portions of the database are stored in multiple physical locations, and in which processing is dispersed or replicated among different points in a network.



Distributed databases can be homogeneous or heterogeneous. All the physical locations in a homogeneous distributed database system have the same underlying hardware and run the same operating systems and database applications. The hardware, operating systems or database applications in a heterogeneous distributed database may be different at each of the locations.

III. CLOUD DATABASE

A cloud database is a database that has been optimized or built for a virtualized environment, either in a hybrid cloud, public cloud or private cloud. Cloud databases provide benefits such as the ability to pay for storage capacity and bandwidth on a per-use basis, and they provide scalability on demand, along with high availability.

A cloud database also gives enterprises the opportunity to support business applications in a software-as-a-service deployment.

IV. OBJECT-ORIENTED DATABASE

Items created using object-oriented programming languages are often stored in relational databases, but object-oriented databases are well-suited for those items.

An object-oriented database is organized around objects rather than actions, and data rather than logic. For example, a multimedia record in a relational database can be a definable data object, as opposed to an alphanumeric value.

V. GRAPH DATABASE

A graph-oriented database, or graph database, is a type of NoSQL database that uses graph theory to store, map and query relationships. Graph databases are basically collections of nodes and edges, where each node represents an entity, and each edge represents a connection between nodes.



Graph databases are growing in popularity for analyzing interconnections. For example, companies might use a graph database to mine data about customers from social media.

1.5. DATABASE LANGUAGES

A database system provides three different types of languages:

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Transaction Control Language (TCL)

1.5.1. DATA DEFINITION LANGUAGE (DDL)

Data-Definition Language is used to specify a database schema by a set of definitions expressed by a special language called a data-definition language (DDL).

For instance, the following statement in the SQL language defines the account table: create table account (account-number char(10), balance integer) Execution of the above DDL statement creates the account table.

In addition, it updates a special set of tables called the data dictionary or data directory. A data dictionary contains metadata—that is, data about data. The schema of a table is an example of metadata. A database system consults the data dictionary before reading or modifying actual data.

We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language. These statements define the implementation details of the database schemas, which are usually hidden from the users.

The data values stored in the database must satisfy certain consistency constraints. For example, suppose the balance on an account should not fall below \$100. The DDL provides facilities to specify such constraints. The database systems check these constraints every time the database is updated.



1.5.2. DATA MANIPULATION LANGUAGE (DML)

Data manipulation is the retrieval of information stored in the database.

- The insertion of new information into the database
- The deletion of information from the database
- The modification of information stored in the database
- The selection of information stored in the database

A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model.

There are basically two types:

- i. Procedural DMLs require a user to specify what data are needed and how to get those data.
- ii. Declarative DMLs (also referred to as nonprocedural DMLs) require a user to specify what data are needed without specifying how to get those data. Declarative DMLs are usually easier to learn and use than are procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data. The DML component of the SQL language is nonprocedural.

A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language. Although technically incorrect, it is common practice to use the terms query language and data manipulation language synonymously.

This query in the SQL language finds the name of the customer whose customer-id is 192-83-7465:

```
select customer.customer-name from customer where customer.customer-id = 192-83-7465;
```

The query specifies that those rows from the table customer where the customer-id is 192-83-7465 must be retrieved, and the customer-name attribute of these rows must be displayed. Queries may involve information from more than one table.

For instance, the following query finds the balance of all accounts owned by the customer with customerid 192-837465.

```
select account.balance from depositor, account where depositor.customer-id = 192-83-7465 and depositor.account-number = account.account-number;
```



There are a number of database query languages in use, either commercially or experimentally.

The levels of abstraction apply not only to defining or structuring data, but also to manipulating data. At the physical level, we must define algorithms that allow efficient access to data. At higher levels of abstraction, we emphasize ease of use.

The goal is to allow humans to interact efficiently with the system. The query processor component of the database system translates DML queries into sequences of actions at the physical level of the database system.

1.6. ACTIONABLE FOR DBA

A **Database Administrator, Database Analyst or Database Developer** is the person **responsible for managing the information** within an organization.

The DBA has many different responsibilities, but the overall goal of the DBA is to **keep the server up at all times** and to provide users with access to the required information when they need it. The DBA makes sure that the **database is protected and that any chance of data loss is minimized**.

A **DBA can be a programmer** who, by default or by volunteering, took over the responsibility of **maintaining a SQL Server** during project development and enjoyed the job so much that he switched.

A **DBA can be a system administrator** who was given the added responsibility of maintaining a SQL Server. DBAs can even come from unrelated fields, such as accounting or the help desk, and switch to Information Systems to become DBAs.

DBA RESPONSIBILITIES

The following are the responsibilities of the database administrator:

I. INSTALLING AND UPGRADING AN SQL SERVER



The DBA is responsible for installing SQL Server or upgrading an existing SQL Server. In the case of upgrading SQL Server, the DBA is responsible for ensuring that if the upgrade is not successful, the SQL Server can be rolled back to an earlier release until the upgrade issues can be resolved.

II. USING STORAGE PROPERLY

SQL Server 2000 enables you to automatically grow the size of your databases and transaction logs, or you can choose to select a fixed size for the database and transaction log. Either way, maintaining the proper use of storage means monitoring space requirements and adding new storage space (disk drives) when required.

III. PERFORMING BACKUP AND RECOVERY DUTIES

Backup and recovery are the DBA's most critical tasks; they include the following aspects:

- Establishing standards and schedules for database backups
- Developing recovery procedures for each database
- Making sure that the backup schedules meet the recovery requirements

IV. MANAGING DATABASE USERS AND SECURITY

With SQL Server 2000, the DBA works tightly with the Windows NT administrator to add user NT logins to the database. The DBA is also responsible for assigning users to databases and determining the proper security level for each user. Within each database, the DBA is responsible for assigning permissions to the various database objects such as tables, views, and stored procedures.

V. TRANSFERRING DATA

The DBA is responsible for importing and exporting data to and from the SQL Server.

VI. REPLICATING DATA

Managing and setting up replication topologies is a big undertaking for a DBA because of the complexities involved with properly setting up and maintaining replication.

The DBA should possess the following skills

1. A good knowledge of the operating system(s)
2. A good knowledge of physical database design



3. Ability to perform both Oracle and also operating system performance monitoring and the necessary adjustments.
4. Be able to provide a strategic database direction for the organization.
5. Excellent knowledge of Oracle backup and recovery scenarios.
6. Good skills in all Oracle tools.
7. A good knowledge of Oracle security management.
8. A good knowledge of how Oracle acquires and manages resources.
9. Sound knowledge of the applications at your site.
10. Experience and knowledge in migrating code, database changes, data and
11. Menus through the various stages of the development life cycle.
12. A good knowledge of the way Oracle enforces data integrity.
13. A sound knowledge of both database and program code performance tuning.
14. A DBA should possess a sound understanding of the business.
15. A DBA should have sound communication skills with management, development teams, vendors, systems administrators and other related service providers.

REVIEW QUESTIONS

1. Define Database Management Systems.
2. Why we need integrity constraints?
3. Illustrate transaction properties.
4. Differentiate volatile and non-volatile storage.
5. What is Data Base Administrator? Discuss the functions of DBA.
6. Explain DBMS applications.
7. What is the instance of a relation?
8. Discuss abstract view of data with diagram.
9. List various types of database users. Explain.
10. Explain briefly the languages supported by database systems.



11. Give examples of systems in which it may make sense to use traditional file processing instead of a database approach.
12. Discuss the role of a high-level data model in the database design process.

OBJECTIVE TYPE QUESTIONS

1. A Database Management System (DBMS) is

- a) Collection of interrelated data
- b) Collection of programs to access data
- c) Collection of data describing one particular enterprise
- d) All of the above

2. A database management software (DBMS) includes

- a) Automated tools (CASE) used to design databases and application programs
- b) A software application that is used to define, create, maintain and provide controlled access to user databases
- c) Application programs that are used to provide information to users
- d) Database that contains occurrences of logically organised data or information

3. Making a change to the conceptual schema of a database but not affecting the existing external schemas is an example of

- (a) Physical data independence
- (b) Concurrency Control
- (c) Logical data independence
- (d) Functional dependency

4. Which of the following is not a level of data abstraction?

- a) Physical Level
- b) Critical Level
- c) Logical Level
- d) View Level

5. Disadvantages of File systems to store data is:

- a) Data redundancy and inconsistency
- b) Difficulty in accessing data
- c) Data isolation
- d) All of the above



6. In an Entity-Relationship Diagram Rectangles represents

- a) Entity sets
- b) Attributes
- c) Database
- d) Tables

7. Which of the following is not a Storage Manager Component?

- a) Transaction Manager
- b) Logical Manager
- c) Buffer Manager
- d) File Manager

8. Data Manipulation Language enables users to

- a) Retrieval of information stored in database
- b) Insertion of new information into the database
- c) Deletion of information from the database
- d) All of the above

9. Which of the following is not an Schema?

- a) Database Schema
- b) Physical Schema
- c) Critical Schema
- d) Logical Schema

10. Which of the following is Database Language?

- a) Data Definition Language
- b) Data Manipulation Language
- c) Query Language
- d) All of the above

11. Which of the following in not a function of DBA?

- a) Network Maintenance
- b) Routine Maintenance
- c) Schema Definition
- d) Authorization for data access

12. Snapshot of the data in the database at a given instant of time is called

- a) Database Schema



- b) Database Instance
- c) Database Snapshot
- d) All of the above

13. Data Manipulation Language (DML) is not to

- a) Create information table in the Database
- b) Insertion of new information into the Database
- c) Deletion of information in the Database
- d) Modification of information in the Database

14. Which of the following is an unary operation?

- a) Selection operation
- b) Generalized selection
- c) Primitive operation
- d) Projection operation

15. Which of the following is the structure of the Database?

- a) Table
- b) Schema
- c) Relation
- d) None of these

16. The DBMS language component which can be embedded in a program is

- a) The data definition language (DDL).
- b) The data manipulation language (DML).
- c) The database administrator (DBA).
- d) A query language.

17. A data dictionary is a special file that contains:

- a) The name of all fields in all files.
- b) The width of all fields in all files.
- c) The data type of all fields in all files.
- d) All of the above.

18. A relational database developer refers to a record as

- a) A Criteria.
- b) A Relation.
- c) A Tuple.
- d) An Attribute.



- 19. Transaction processing is associated with everything below except**
- Producing detail, summary, or exception reports.
 - Recording a business activity.
 - Confirming an action or triggering a response.
 - Maintaining data.
- 20. It is possible to define a schema completely using**
- VDL and DDL.
 - DDL and DML.
 - SDL and DDL.
 - VDL and DML.
- 21. The method of access which uses key transformation is known as**
- Direct.
 - Hash.
 - Random.
 - Sequential.
- 22. Data independence means**
- Data is defined separately and not included in programs.
 - Programs are not dependent on the physical attributes of data.
 - Programs are not dependent on the logical attributes of data.
 - Both (B) and (C).
- 23. The statement in SQL which allows to change the definition of a table is**
- Alter.
 - Update.
 - Create.
 - select
- 24. _____ is a classical approach to database design?**
- Left – Right approach
 - Right – Left approach
 - Top – Down approach
 - Bottom – Up approach
- 25. _____ refers to the correctness and completeness of the data in a database?**
- Data security
 - Data integrity
 - Data constraint



d) Data independence

26. A table that displays data redundancies yields _____ anomalies.

- a) Insertion
- b) Deletion
- c) Update
- d) All of the above

27. The database schema is written in

- a) HLL
- b) DML
- c) DDL
- d) DCL

28. In the architecture of a database system external level is the

- a) Physical level.
- b) Logical level.
- c) Conceptual level
- d) View level.

29. A collection of conceptual tools for describing data, relationships, semantics and constraints is referred to as _____

- a) Data Model
- b) E-R Model
- c) DBMS
- d) All of the above

30. Dr.E.F. Codd represented rules that a database must obey if it has to be considered truly relational.

- a) 10
- b) 15
- c) 14
- d) 12

31. The language used in application programs to request data from the DBMS is referred to as the _____

- a) DML
- b) DDL
- c) VDL
- d) SDL



32. A logical schema

- a) is the entire database.
- b) is a standard way of organizing information into accessible parts.
- c) describes how data is actually stored on disk.
- d) both (A) and (C)

33. Related fields in a database are grouped to form a

- a) data file.
- b) data record.
- c) Menu. (d) bank.

34. The database environment has all of the following components except:

- a) users.
- b) separate files.
- c) database.
- d) database administrator.

35. The language which has recently become the defacto standard for interfacing application programs with relational database system is

- a) Oracle.
- b) SQL.
- c) DBase.
- d) 4GL

36. The way a particular application views the data from the database that the application uses is a

- a) module.
- b) relational model.
- c) schema.
- d) sub schema.

37. Which of the following is not correct?

- a) Each entity must include some descriptive information
- b) If an object only requires an identifier, it should be classified as an attribute
- c) Each multivalued attribute should be classified as an entity even if it does not have any descriptive information
- d) The procedure of identifying entities and attaching attributes always leads to a unique solution

38. In the relational modes, cardinality is termed as:

- a) Number of tuples.
- b) Number of attributes.
- c) Number of tables.



d) Number of constraints.

39. Relational calculus is a

- a) Procedural language.
- b) Non- Procedural language.
- c) Data definition language.
- d) High level language.

40. The view of total database content is

- a) Conceptual view.
- b) Internal view.
- c) External view.
- d) Physical View.

41. Which one of the following statements is false?

- a) The data dictionary is normally maintained by the database administrator.
- b) Data elements in the database can be modified by changing the data dictionary.
- c) The data dictionary contains the name and description of each data element.
- d) The data dictionary is a tool used exclusively by the database administrator.

42. An advantage of the database management approach is

- a) Data is dependent on programs.
- b) Data redundancy increases.
- c) Data is integrated and can be accessed by multiple programs.
- d) None of the above.

43. A DBMS query language is designed to

- a) Support end users who use English-like commands.
- b) Support in the development of complex applications software.
- c) Specify the structure of a database.
- d) All of the above.

44. A relationship between the instances of a single entity type is called a ___relationship.

- a) Ternary
- b) Primary
- c) Binary
- d) Unary.

45. The users who use easy-to-use menu are called

- a) Sophisticated end users.
- b) Naïve users.
- c) Stand-alone users.
- d) Casual end users.



திருவள்ளூர் பல்கலைக்கழகம்
THIRUVALLUVAR UNIVERSITY

(State University Accredited with "B" Grade by NAAC)
Serkkadu, Vellore - 632 115, Tamil Nadu, India.

E-NOTES / CS & BCA

KEYS

1-a, 2-b, 3-c, 4-b, 5-d, 6-a, 7-b, 8-d, 9-c, 10-d, 11-a, 12-b, 13-a,
14-b, 15-b, 16-b, 17-d, 18-c, 19-c, 20-b, 21-b, 22-d, 23-a, 24-c, 25-b, 26-d,
27-c, 28-d, 29-a, 30-d, 31-a, 32-a, 33-b, 34-a, 35-b, 36-d, 37-d, 38-a, 39-b,
40-a, 41-b, 42-c, 43-d, 44-d, 45-b.



CHAPTER – 2: THE ENTITY – RELATIONSHIP MODEL

2.1. INTRODUCTION

An entity is any object, place or activity about which an enterprise keeps data. It is an object which can have instances or occurrences. Each instance should be capable of being uniquely identified.

Entity type and entity instance are two important terms related to entities. An entity type is a set of objects which share common properties. An entity type could be employee or student or teacher.

The Database Design consists of three components:

- i. Conceptual Design
- ii. Data Modeling (Entity-Relationship Diagrams and Normalization)
- iii. Physical Design and Implementation.

The scope of E-R model includes:

- Entity and entity sets and reducing E-R diagram to tables
- Relationship and relationship sets
- Attributes
- Mapping Constraints
- Keys
- Primary keys for relationship sets
- Entity relationship diagrams
- Representation of strong entity sets
- Representation of weak entity sets
- Representation of relationship sets
- Generalization
- Aggregation

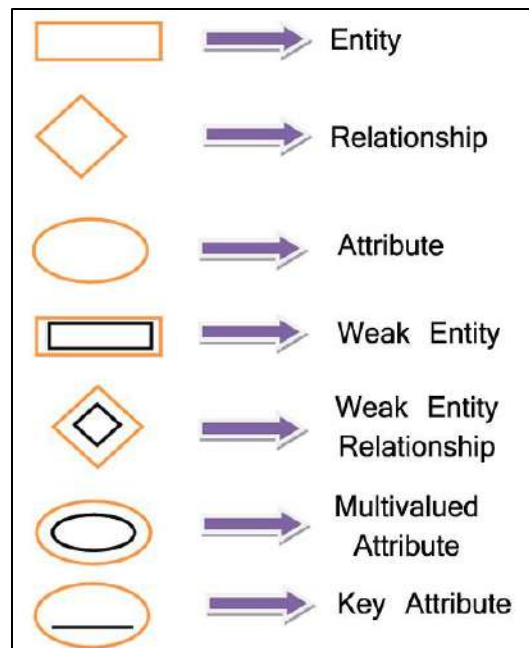


2.2. THE ENTITY RELATIONSHIP MODEL

E-R diagram is the short form of "Entity-Relationship" diagram. An E-R diagram efficiently shows the relationships between various entities stored in a database. E-R diagrams are used to model real-world objects like a person, a car, a company etc. and the relation between these real-world objects.

An E-R diagram has following features:

- E-R diagrams are used to represent E-R model in a database, which makes them easy to be converted into relations (tables).
- E-R diagrams provide the purpose of real-world modeling of objects which makes them intently useful.
- E-R diagrams require no technical knowledge & no hardware support.
- These diagrams are very easy to understand and easy to create even by a naive user.
- It gives a standard solution of visualizing the data logically.






2.2.1. COMPONENTS OF AN E-R DIAGRAM


An E-R diagram constitutes of following Components

A. Entity:- Any real-world object can be represented as an entity about which data can be stored in a database. All the real world objects like a book, an organization, a product, a car, a person are the examples of an entity.

Any living or non-living objects can be represented by an entity. An entity is symbolically represented by a rectangle enclosing its name.

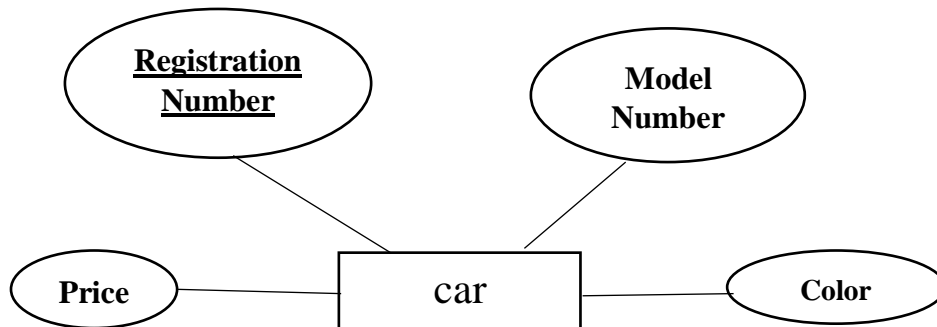
Entities can be characterized into two types:

- **Strong entity:** A strong entity has a primary key attribute which uniquely identifies each entity. Symbol of strong entity is  same as an entity.

- **Weak entity:** A weak entity  does not have a primary key attribute and depends on other entity via a foreign key attribute.



B. Attribute:- Each entity has a set of properties. These properties of each entity are termed as attributes. For example, a car entity would be described by attributes such as price, registration number, model number, color etc. Attributes are indicated by ovals in an E-R diagram. A primary key attribute is depicted by an underline in the E-R diagram.



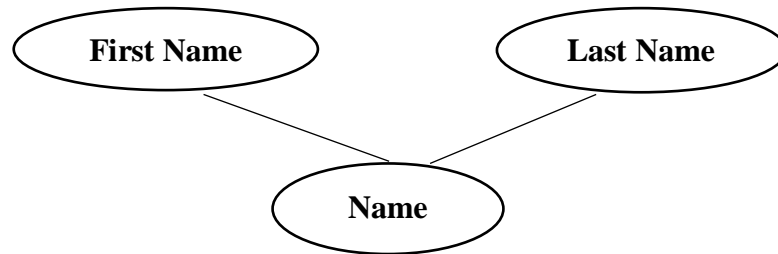
An attribute can be characterized into following types:



- **Simple attribute**:- An attribute is classified as a simple attribute if it cannot be partitioned into smaller components. For example, age and sex of a person. A simple attribute is represented by an oval.



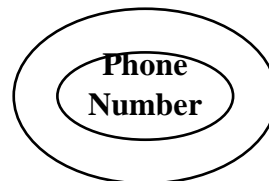
- **Composite attribute**:- A composite attribute can be subdivided into smaller components which further form attributes. For example, 'name' attribute of an entity "person" can be broken down into first name and last name which further form attributes. Grouping of these related attributes forms a composite attribute. 'name' is the composite attribute in this example.



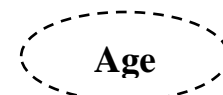
- **Single valued attribute**:- If an attribute of a particular entity represents single value for each instance, then it is called a single-valued attribute. For example, Ramesh, Kamal and Suraj are the instances of entity 'student' and each of them is issued a separate roll number. A single oval is used to represent this attribute.



- **Multi valued attribute**:- An attribute which can hold more than one value, it is then termed as multi-valued attribute. For example, phone number of a person. Symbol of multi-valued attribute is shown below,



- **Derived attribute**: A derived attribute calculate its value from another attribute. For example, 'age' is a derived attribute if it calculates its value from 'current date' & 'birth date' attributes. A derived attribute is represented by a dashed oval.



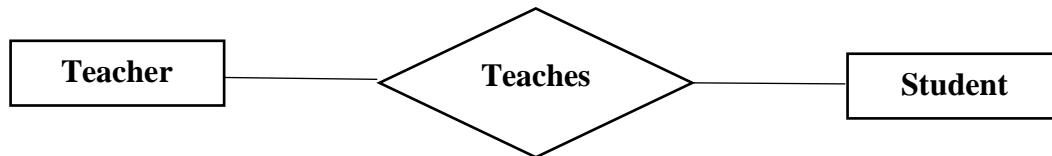


C. Relationships:- A relationship is defined as bond or attachment between 2 or more entities. Normally, a verb in a sentence signifies a relationship.

For example,

- An employee assigned a project.
- Teacher teaches a student.
- Author writes a book.

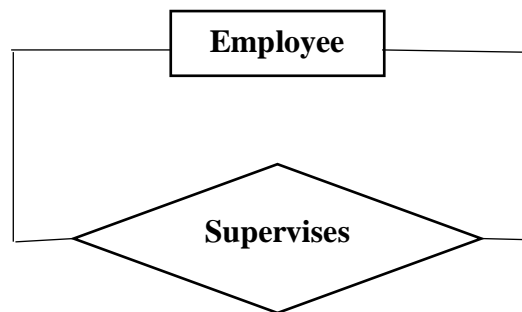
A diamond is used to symbolically represent a relationship in the e-r diagram.



2.2.2. VARIOUS TERMS RELATED TO RELATIONSHIPS

a). Degree of relationship:- It signifies the number of entities involved in a relationship. Degree of a relationship can be classified into following types:

- **Unary relationship:-** If only single entity is involved in a relationship then it is a unary relationship. For example, An employee(manager) supervises another employee.



- **Binary relationships:-** when two entities are associated to form a relation, then it is known as a binary relationship. For example, A person works in a company. Most of the times we use only binary relationship in an e-r diagram. The teacher-student example shown above signifies a binary relationship.

Other types of relationships are ternary and quaternary. As the name signifies, a ternary relationship is associated with three entities and a quaternary relationship is associated with four entities.



b.) Connectivity of a relationship:- Connectivity of a relationship describes, how many instances of one entity type are linked to how many instances of another entity type.

Various categories of connectivity of a relationship are:

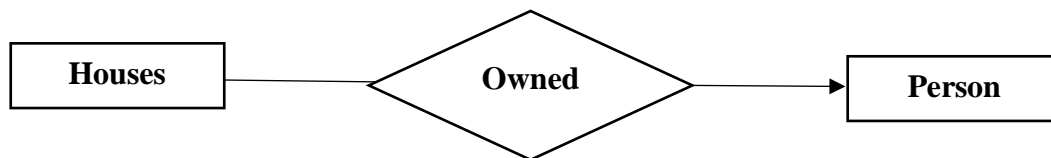
- **One to One (1:1)** – “Student allotted a project” signifies a one-to-one relationship because only one instance of an entity is related with exactly one instance of another entity type.



- **One to Many (1:M)** – “A department recruits faculty” is a one-to-many relationship because a department can recruit more than one faculty, but a faculty member is related to only one department.



- **Many to One (M:1)** – “Many houses are owned by a person” is a many-to-one relationship because a person can own many houses but a particular house is owned only a person.



- **Many to Many (M:N)** – “Author writes books” is a many-to-many relationship because an author can write many books and a book can be written by many authors.



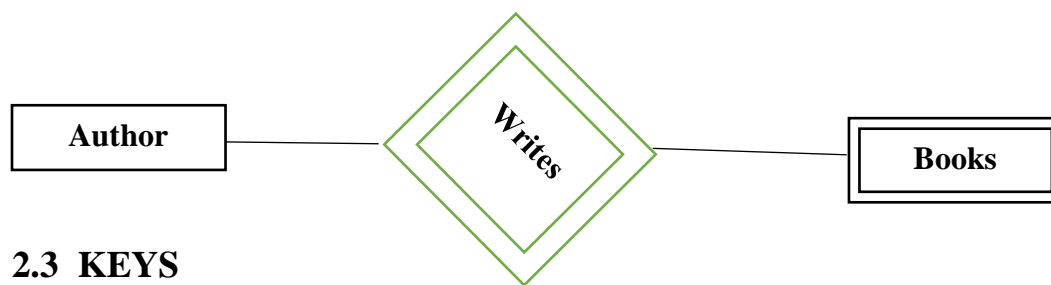
c) Weak Entity Type and Identifying Relationship:

An entity type has a key attribute which uniquely identifies each entity in the entity set. But there exists **some entity type for which key attribute can't be defined**. These are called Weak Entity type.



For example, A company may store the information of dependants (Parents, Children, Spouse) of an Employee. But the dependents don't have existence without the employee. So Dependent will be weak entity type and Employee will be Identifying Entity type for Dependant.

A weak entity type is represented by a double rectangle. The participation of weak entity type is always total. The relationship between weak entity type and its identifying strong entity type is called identifying relationship and it is represented by double diamond.



2.3 KEYS

- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

For example: In Student table, ID is used as a key because it is unique for each student. In PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.

2.3.1. TYPES OF KEY

1. Primary key

- It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys as we saw in PERSON table. The key which is most suitable from those lists become a primary key.
- In the EMPLOYEE table, ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary key since they are also unique.
- For each entity, selection of the primary key is based on requirement and developers.



2. Candidate key

- A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
- The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.

3. Super Key

Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.

For example: In the above EMPLOYEE table, for(EMPLOYEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLOYEE_ID can't be the same. Hence, this combination can also be a key.

The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

4. Foreign key

- Foreign keys are the column of the table which is used to point to the primary key of another table.
- In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



REVIEW QUESTIONS

1. What are the major components used in E-R diagram design?
2. How do we represent null values? Discuss the importance of handling null values.
3. How to maintain class hierarchies in ER-Diagrams? Explain with employee database.
4. Explain the following terms:
 - (i) Entity and entity set.
 - (ii) Attribute and attribute sets.
 - (iii) Relationship and relationship sets.
5. Differentiate between primary key and a candidate key.
6. Why foreign key constraints are important? Explain with employee database.
7. What is the difference between a key and a super key?
8. Why do we designate one of the candidate keys of a relation to be the primary key?
9. Discuss the characteristics of relations that make them different from ordinary tables and files.
10. Discuss the various reasons that lead to the occurrence of NULL values in relations.
11. Discuss the entity integrity and referential integrity constraints. Why each is considered important?
12. Define *foreign key*. What is this concept used for?
13. How can the key and foreign key constraints be enforced by the DBMS? Is the enforcement technique you suggest difficult to implement? Can the constraint checks be executed efficiently when updates are applied to the database?

OBJECTIVE TYPE QUESTIONS

- 1. The minimal set of super key is called**
 - a) Primary key
 - b) Secondary key
 - c) Candidate key
 - d) Foreign key
- 2. An entity set that does not have sufficient attributes to form a primary key is a**
 - a) Strong entity set.
 - b) Weak entity set.
 - c) Simple entity set.
 - d) Primary entity set.
- 3. Minimal Super keys are called**
 - a) Schema keys
 - b) Candidate keys
 - c) Domain keys
 - d) Attribute keys



4. **The Primary key must be**
 - a) Non Null
 - b) Unique
 - c) Option A or B
 - d) Option A and B

5. **The attribute that can be divided into other attributes is called**
 - a) Simple Attribute
 - b) Composite Attribute
 - c) Multi-valued Attribute
 - d) Derived Attribute

6. **In an Entity-Relationship Diagram "Ellipses" represents**
 - a) Attributes
 - b) Weak entity set
 - c) Relationship sets
 - d) Multi-valued attributes

7. **In an Entity-Relationship Diagram "Diamonds" represents**
 - a) Attributes
 - b) Multi-valued attributes
 - c) Weak entity set
 - d) Relationship sets

8. **Which of the following is true regarding Referential Integrity?**
 - a) Every primary-key value must match a primary-key value in an associated table
 - b) Every primary-key value must match a foreign-key value in an associated table
 - c) Every foreign-key value must match a primary-key value in an associated table
 - d) Every foreign-key value must match a foreign-key value in an associated table

9. **How many types of keys in Database Design?**
 - a) Candidate key
 - b) Primary key
 - c) Foreign key
 - d) All of these

10. **An entity type whose existence depends on another entity type is called a _____ entity.**
 - a) Strong
 - b) Weak
 - c) Codependent
 - d) Independent

11. **Key to represent relationship between tables is called**
 - a) Primary key
 - b) Secondary
 - c) Foreign Key
 - d) None of these

12. **When converting one (1) to many (N) binary relationship into tables, the recommended solution is usually**
 - a) One big table with all attributes from both entities included
 - b) Foreign key added on the Child (many side) referencing the parent
 - c) Foreign key added on the Parent (one side) referencing the child
 - d) Foreign key added on both sides (both tables)



13. The relational model feature is that there

- a) Is no need for primary key data.
- b) Is much more data independence than some other database models.
- c) Are explicit relationships among records.
- d) Are tables with many dimensions.

14. Conceptual design

- a) Is a documentation technique.
- b) Needs data volume and processing frequencies to determine the size of the database.
- c) Involves modelling independent of the DBMS.
- d) Is designing the relational model.

15. The method in which records are physically stored in a specified order according to a key field in each record is

- a) Hash.
- b) Direct.
- c) Sequential.
- d) All of the above

16. A subschema expresses

- a) the logical view.
- b) the physical view.
- c) the external view.
- d) all of the above.

KEYS

1-c, 2-b, 3-b, 4-d, 5-b, 6-a, 7-d, 8-c, 9-d, 10-b, 11-c, 12-b, 13-b, 14-c, 15-a, 16-c.



CHAPTER – 3 : DATA MODELS

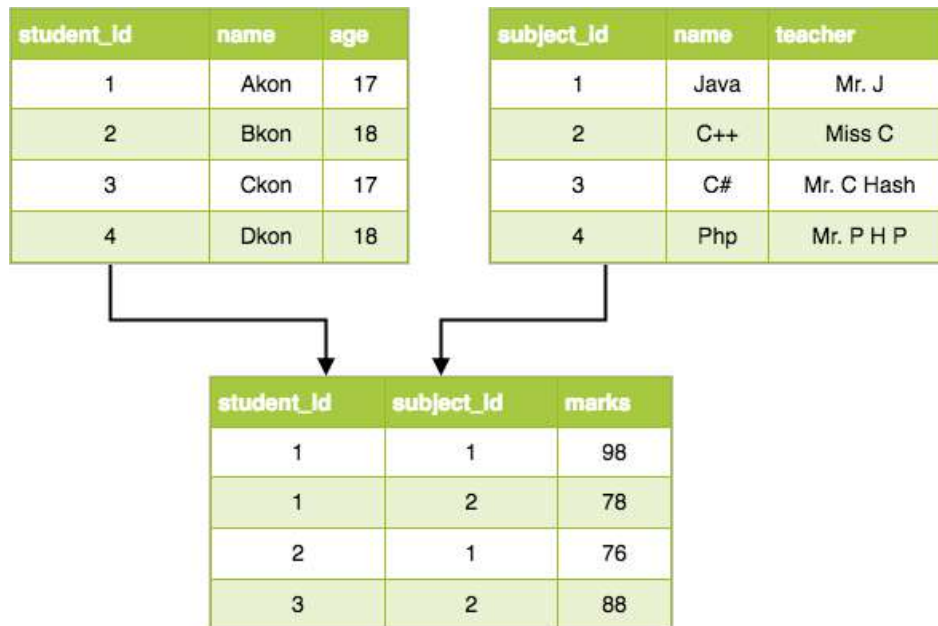
3.1. INTRODUCTION

A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system. While the **Relational Model** is the most widely used database model, there are other models too:

3.2. TYPES OF DATA MODELS

- i. Relational Model
- ii. Entity-relationship Model
- iii. Hierarchical Model
- iv. Network Model

i. RELATIONAL MODEL



In this model, data is organised in two-dimensional **tables** and the relationship is maintained by storing a common field. This model was introduced by E.F Codd in 1970, and since then it has



been the most widely used database model, infact, we can say the only database model used around the world.

The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table. Hence, tables are also known as **relations** in relational model.

CODD'S 12 RULES FOR RELATIONAL DATABASE

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

RULE 1: INFORMATION RULE

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

RULE 2: GUARANTEED ACCESS RULE

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

RULE 3: SYSTEMATIC TREATMENT OF NULL VALUES

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.



RULE 4: ACTIVE ONLINE CATALOG

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

RULE 5: COMPREHENSIVE DATA SUB-LANGUAGE RULE

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

RULE 6: VIEW UPDATING RULE

All the views of a database, which can theoretically be updated, must also be updatable by the system.

RULE 7: HIGH-LEVEL INSERT, UPDATE, AND DELETE RULE

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

RULE 8: PHYSICAL DATA INDEPENDENCE

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

RULE 9: LOGICAL DATA INDEPENDENCE

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or



one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

RULE 10: INTEGRITY INDEPENDENCE

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

RULE 11: DISTRIBUTION INDEPENDENCE

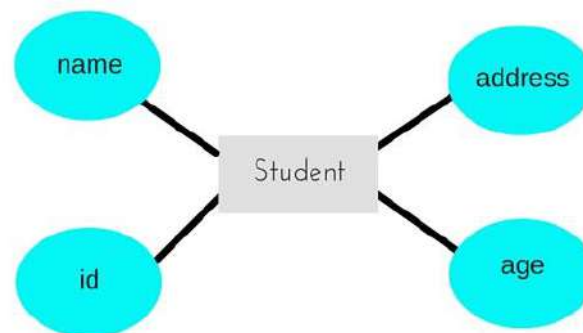
The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

RULE 12: NON-SUBVERSION RULE

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints

ii. ENTITY-RELATIONSHIP MODEL

In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes. Different Entities are related using relationships. E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand.

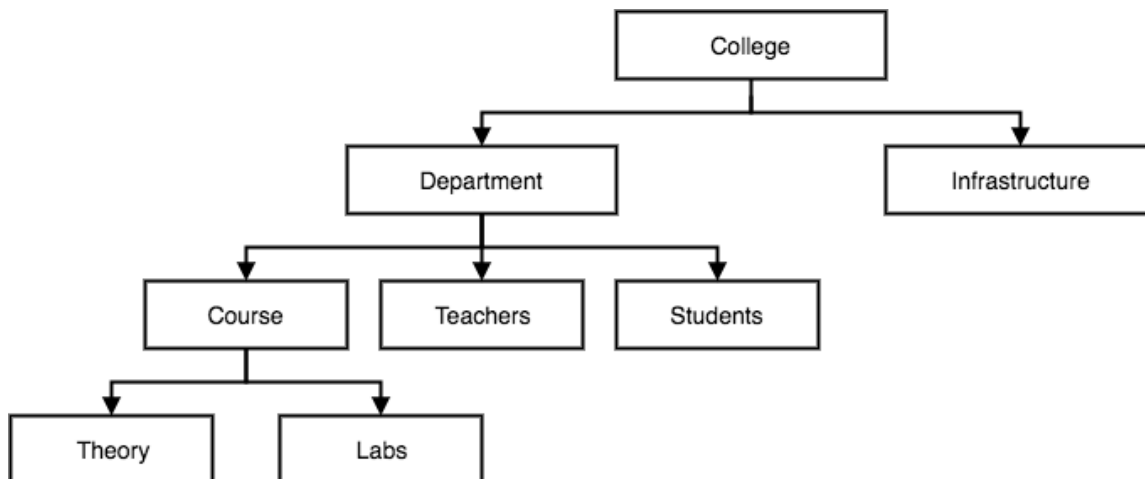




This model is good to design a database, which can then be turned into tables in relational model(explained below). Let's take an example, If we have to design a School Database, then **Student** will be an **entity** with **attributes** name, age, address etc. As **Address** is generally complex, it can be another **entity** with **attributes** street name, pincode, city etc, and there will be a relationship between them.

iii.HIERARCHICAL MODEL

This database model organizes data into a tree-like-structure, with a single root, to which all the other data is linked. The hierarchy starts from the **Root** data, and expands like a tree, adding child nodes to the parent nodes. In this model, a child node will only have a single parent node. This model efficiently describes many real-world relationships like index of a book, recipes etc.



In hierarchical model, data is organized into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.

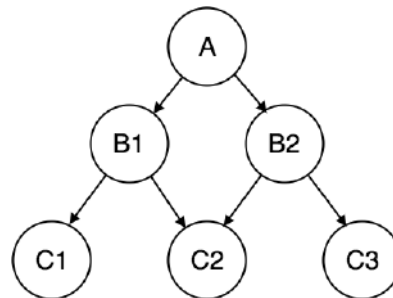


iv.NETWORK MODEL

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before Relational Model was introduced.



REVIEW QUESTIONS

1. What is Data modeling? Explain relational model.
2. Explain about Entity-Relationship model with an example.
3. Consider the following set of requirements for a UNIVERSITY database that is used to keep track of students' transcripts. This is similar but not identical to the database shown in Figure 1:
 - a. The university keeps track of each student's name, student number, Social Security number, current address and phone number, permanent address and phone number, birth date, sex, class (freshman, sophomore, ..., graduate), major department, minor department (if any), and degree program (B.A., B.S., ..., Ph.D.). Some user applications need to refer to the city, state, and ZIP Code of the student's permanent address and to the student's last name. Both Social Security number and student number have unique values for each student.



STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

Figure 1:

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310



- b. Each department is described by a name, department code, office number, office phone number, and college. Both name and code have unique values for each department.
 - c. Each course has a course name, description, course number, number of semester hours, level, and offering department. The value of the course number is unique for each course.
 - d. Each section has an instructor, semester, year, course, and section number. The section number distinguishes sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, ..., up to the number of sections taught during each semester.
 - e. A grade report has a student, section, letter grade, and numeric grade (0, 1, 2, 3, or 4).
- Design an ER schema for this application, and draw an ER diagram for the schema. Specify key attributes of each entity type, and structural constraints on each relationship type. Note any unspecified requirements, and make appropriate assumptions to make the specification complete.
4. A database is being constructed to keep track of the teams and games of a sports league. A team has a number of players, not all of whom participate in each game. It is desired to keep track of the players participating in each game for each team, the positions they played in that game, and the result of the game. Design an ER schema diagram for this application, stating any assumptions you make. Choose your favorite sport (e.g., soccer, baseball, football).

OBJECTIVE TYPE QUESTIONS

1. Which of the following is a Data Model?

- a) Entity-Relationship model
- b) Relational data model
- c) Object-Based data model
- d) All of the above

2. Who proposed the relational model?

- a) Bill Gates
- b) E.F. Codd
- c) Herman Hollerith
- d) Charles Babbage



3. In an E-R diagram attributes are represented by

- a) Rectangle.
- b) Square.
- c) Ellipse.
- d) Triangle.

4. E-R model uses this symbol to represent weak entity set ?

- a) Dotted rectangle.
- b) Diamond
- c) Doubly outlined rectangle
- d) None of these

5. SET concept is used in :

- a) Network Model
- b) Hierarchical Model
- c) Relational Model
- d) None of these

6. Which of the following is record based logical model?

- a) Network Model
- b) Object oriented model
- c) E-R Model
- d) None of these

KEYS

1-d, 2-b, 3-c, 4-c, 5-a, 6-a.



UNIT – II

CHAPTER 4: RELATIONAL ALGEBRA

4.1. INTRODUCTION

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries. It collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform this action.

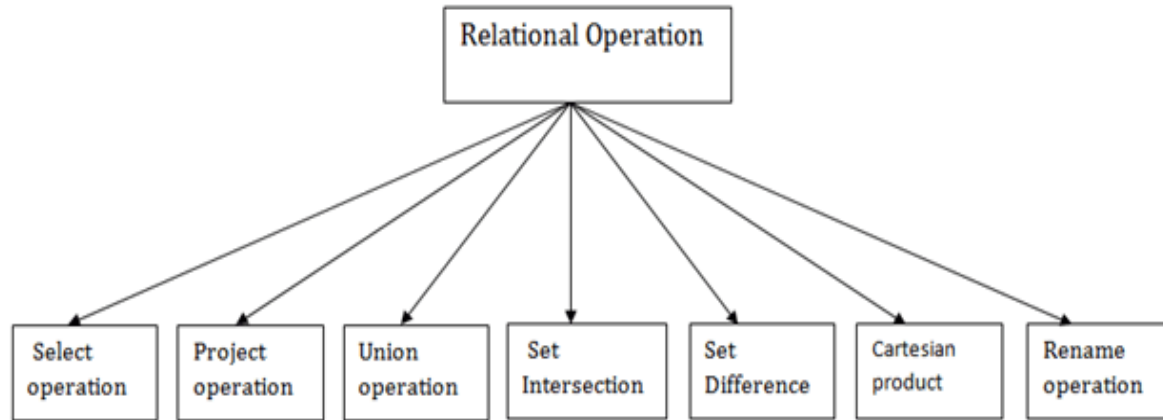
Relational algebra operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input

- A relational database consists of a collection of **tables**, each having a unique **name**. A **row** in a table represents a **relationship** among a set of values. Thus a table represents a **collection of relationships**.
- There is a direct correspondence between the concept of a table and the mathematical concept of a relation. A substantial theory has been developed for relational databases.
- Procedural language
- Six basic operators
 1. select: σ
 2. project: Π
 3. union: \cup
 4. set difference: $-$
 5. Cartesian product: \times
 6. rename: ρ

The operators take one or two relations as inputs and produce a new relation as a result



4.2. STRUCTURE OF RELATIONAL DATABASES



4.2.1. SELECT OPERATION

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ) Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operation selects tuples that satisfy a given predicate.

Notation: $\sigma p(r)$

Where:

σ is used for selection prediction.

r is used for relation.

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

For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Trichy	L-17	1000
Salem	L-23	2000
Vellore	L-15	1500
Trichy	L-14	1500
Erode	L-13	500



Madurai	L-11	900
Vellore	L-16	1300

QUERY: σ BRANCH_NAME="Vellore" (LOAN)

OUTPUT:

BRANCH_NAME	LOAN_NO	AMOUNT
Vellore	L-15	1500
Vellore	L-16	1300

4.2.2. PROJECT OPERATION

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminate duplicate values. (Π) The symbol used to choose attributes from a relation. This operation helps you to keep specific columns from a relation and discards the other columns.

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

Notation: Π A1, A2, An (r)

Where

A1, A2, A3 is used as an attribute name of relation r.

Example: CUSTOMER RELATION

NAME	STREET	CITY
Selvam	Main	Chennai
Kumar	North	Harur



Hari	Main	Chennai
Kavin	North	Harur
Jothi	Alma	Coimbatore
Balaji	Senator	Coimbatore

QUERY: [] NAME, CITY (CUSTOMER)

OUTPUT:

NAME	CITY
Selvam	Chennai
Kumar	Harur
Hari	Chennai
Kavin	Harur
Jothi	Coimbatore
Balaji	Coimbatore

4.2.3. UNION OPERATION

Union operation in relational algebra is same as union operation in set theory, only constraint is for union of two relation both relation must have same set of Attributes.

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by \cup .

Notation: $R \cup S$

A union operation must hold the following condition:

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



Example:DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Jothi	A-101
Kumar	A-121
Hari	A-321
Tamil	A-176
Jothi	A-273
Selvam	A-472
Arun	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Selvam	L-17
Kumar	L-23
Hari	L-15
Jeni	L-14
Kavin	L-93
Kumar	L-11
Suman	L-17

QUERY: Π CUSTOMER_NAME (BORROW) \cup Π CUSTOMER_NAME (DEPOSITOR)



OUTPUT:

CUSTOMER_NAME
Jothi
Kumar
Hari
Tamil
Selvam
Lindsay
Jeni
Kavin
Suman
Mayes

4.2.4. SET INTERSECTION

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

Notation: $R \cap S$

Example: Using the above DEPOSITOR table and BORROW table

QUERY: \cap CUSTOMER_NAME (BORROW) \cap \cap CUSTOMER_NAME (DEPOSITOR)

OUTPUT:

CUSTOMER_NAME
Kumar
Selvam

4.2.5. SET DIFFERENCE

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.



- It is denoted by intersection minus (-).

Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table

QUERY: Π CUSTOMER_NAME (BORROW) - Π CUSTOMER_NAME (DEPOSITOR)

OUTPUT:

CUSTOMER_NAME
Jeni
Hari
Suman
Kavin

4.2.6. CARTESIAN PRODUCT

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

Notation: E X D

Example:EMPLOYEE RELATION

DEPARTMENT RELATION

EMP_ID	EMP_NAME	EMP_DEPT
1	Kumar	A
2	Hari	C
3	John	B

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

QUERY:EMPLOYEE X DEPARTMENT

OUTPUT:



EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Kumar	A	A	Marketing
1	Kumar	A	B	Sales
1	Kumar	A	C	Legal
2	Hari	C	A	Marketing
2	Hari	C	B	Sales
2	Hari	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

4.2.7. RENAME OPERATION

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

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

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

4.3. FUNDAMENTAL RELATIONAL ALGEBRA OPERATIONS

4.3.1. JOIN OPERATIONS

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by \bowtie .

Example:EMPLOYEE RELATION



EMP_CODE	EMP_NAME
101	Kumar
102	Jeni
103	Hari

SALARY RELATION

EMP_CODE	SALARY
101	50000
102	30000
103	25000

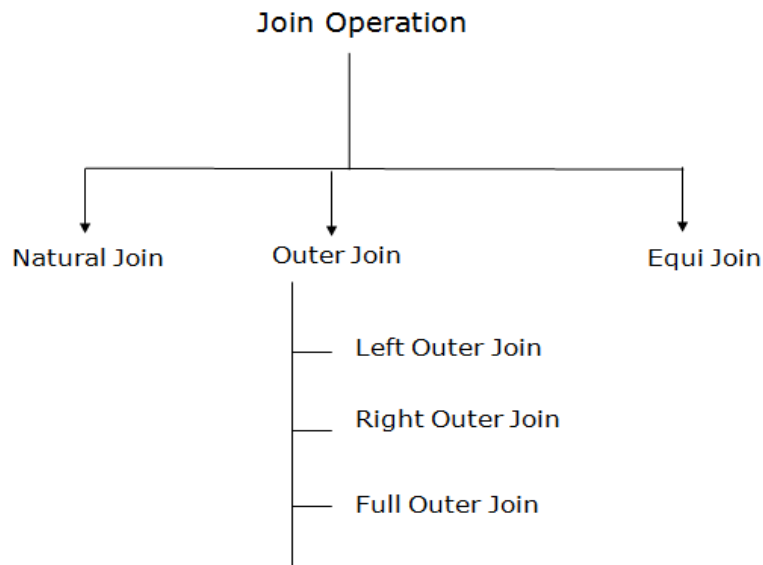
QUERY: (EMPLOYEE ⋈ SALARY)

OUTPUT:

EMP_CODE	EMP_NAME	SALARY
101	Kumar	50000
102	Jeni	30000
103	Hari	25000



4.3.2. TYPES OF JOIN OPERATIONS



(A) NATURAL JOIN

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- It is denoted by \bowtie .

Example: Let's use the above EMPLOYEE table and SALARY table:

QUERY: Π EMP_NAME, SALARY (EMPLOYEE \bowtie SALARY)

OUTPUT:

EMP_NAME	SALARY
Kumar	50000
Jeni	30000
Hari	25000

(B) OUTER JOIN

The outer join operation is an extension of the join operation. It is used to deal with missing information. An extension of the join operation that avoids loss of information.



Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.

Uses *null* values:

- *null* signifies that the value is unknown or does not exist
- All comparisons involving *null* are (roughly speaking) **false** by definition.
- We shall study precise meaning of comparisons with nulls later

Example:EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

QUERY:(EMPLOYEE ⋈ FACT_WORKERS)

OUTPUT:

EMP_NAME	STREET	CITY	BRANCH	SALARY
----------	--------	------	--------	--------



Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

An outer join is basically of three types:

- a. Left outer join
- b. Right outer join
- c. Full outer join

a. Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by \bowtie .

Example: Using the above EMPLOYEE table and FACT_WORKERS table

QUERY:EMPLOYEE \bowtie FACT_WORKERS

OUTPUT:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL



b. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by \bowtie .

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

QUERY:EMPLOYEE \bowtie FACT_WORKERS

OUTPUT:

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

c. Full outer join:

- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- It is denoted by \bowtie .

Example: Using the above EMPLOYEE table and FACT_WORKERS table

QUERY:EMPLOYEE \bowtie FACT_WORKERS



OUTPUT:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

(C) Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

Example: CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Hari
3	Jeni

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

QUERY: CUSTOMER ⋈ PRODUCT



OUTPUT:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Hari	2	Mumbai
3	Hari	3	Noida

4.4. ADDITIONAL RELATIONAL ALGEBRA OPERATIONS

“Additional operations” refer to relational algebra operations that can be expressed in terms of the fundamentals — select, project, union, set-difference, cartesian-product, and rename.

The compositions of these operations are so lengthy, yet so common, that we define new operations for them, based on the fundamentals.

4.4.1. SET-INTERSECTION

The set-intersection operation is a binary operation on relations r and s that is denoted by the traditional intersection symbol, \cap . $r \cap s$ results in all tuples t such that $(t \in r) \wedge (t \in s)$.

Set-intersection is defined in terms of set-difference:

$$r \cap s = r - (r - s)$$

Thus, set-intersection must follow the same compatibility rules as set-difference: same arity, corresponding domains.

4.4.2. THETA JOIN

One can generalize the natural-join operation into a theta join, so named because instead of the specific “attribute-matching” condition involved in natural-join, we allow θ to be any predicate on the attributes in $R \cup S$ for $r(R)$ and $s(S)$.

Thus, we have $r \bowtie_{\theta} s$, defined as



$$r \bowtie s = \sigma_{\theta}(r \times s)$$

4.5. EXTENDED RELATIONAL ALGEBRA OPERATIONS

While they are technically “extensions” to the algebra, they still follow the mathematical rigor and precision that allow us to draw sweeping and powerful conclusions from simpler concepts.

4.5.1. GENERALIZED PROJECTION

The generalized-projection operation extends the fundamental projection operation by allowing arithmetic (or, in the most general case, overall transformative) functions in the projection’s attribute list. It is still denoted with Π , but now the straight-up attribute list A has changed into an expression list F_1, F_2, \dots, F_n

$$\Pi_{F_1, F_2, \dots, F_n}(E)$$

E is any relational algebra expression, which is of course a relation. F_k may be any expression involving constant values and the attributes of E ’s resultant relation schema.

4.6. NULL VALUES

The outer-join operations bring the notion of null values into the relational algebra.

In reality, there are a lot of plausible approaches for handling null. If we stick to the definition of null as an unknown or non-existent value, we can establish the following:

Any arithmetic operations involving null must return null.

Comparing anything to null really doesn’t have much meaning, so we create a new “boolean value” in this case — unknown, meaning that we really can’t say whether a comparison to null is true or false.



By defining the new boolean value unknown, we need to determine how unknown interacts with the other boolean values, true and false, in terms of the Boolean operations \wedge , \vee , and \neg .

4.7. MODIFICATION OF THE DATABASE

So far, all of the operations that we have discussed “derive” new relations from others, but don’t actually modify or alter the original relations. How does one specify operations that change relations “in place?”

The general guideline is to use the assignment operator, but to make assignments to existing relations instead of designating new temporary variables.

4.7.1. DELETION

Deletion of tuples is effectively a set-difference operation that is “permanent.” Thus, we can write deletion as:

$$r \leftarrow r - E$$

E in this case is any relational algebra expression that determines the tuples that are to be removed from r.

4.7.2. INSERTION

Similarly to deletion, insertion of tuples can be viewed as a union operation that is made permanent. Thus, insertion is:

$$r \leftarrow r \cup E$$

E once more is a relational algebra expression that determines the tuples to be setunioned with (thus “inserted into”) r.

4.7.3. UPDATION

An update modification changes one or more values within a tuple. Once more, we find that updating is just a persistent version of another relational operation, this time generalized projection: $r \leftarrow \Pi_{F_1, F_2, \dots, F_n}(r)$.



4.8. TUPLE RELATIONAL CALCULUS

Tuple Relational Calculus is a **non-procedural query language** unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains what to do but not how to do.

In Tuple Calculus, a query is expressed as

$$\{t | P(t)\}$$

where t = resulting tuples,

$P(t)$ = known as Predicate and these are the conditions that are used to fetch t

Thus, it generates set of all tuples t , such that Predicate $P(t)$ is true for t .

$P(t)$ may have various conditions logically combined with OR (\vee), AND (\wedge), NOT(\neg).

It also uses quantifiers:

$\exists t \in r (Q(t))$ = "there exists" a tuple in t in relation r such that predicate $Q(t)$ is true.

$\forall t \in r (Q(t)) = Q(t)$ is true "for all" tuples in relation r .

Example:

Table-1: Customer

CUSTOMER NAME	STREET	CITY
Ram	Gandhi Street	Vellore
Vijay	New Street	Chennai
Chitra	Gandhi Street	Chennai
Ravi	Bazar Street	Bangalore



திருவள்ளூர் பல்கலைக்கழகம்
THIRUVALUVAR UNIVERSITY

(State University Accredited with "B" Grade by NAAC)
Serkkadu, Vellore - 632 115, Tamil Nadu, India.

E-NOTES / CS & BCA

Table-2: Branch

BRANCH NAME	BRANCH CITY
Main	Bangalore
Bazar	Vellore
Market	Chennai

Table-3: Account

ACCOUNT NUMBER	BRANCH NAME	BALANCE
1111	Main	50000
1112	Bazar	10000
1113	Market	9000
1114	Main	7000

Table-4: Loan

LOAN NUMBER	BRANCH NAME	AMOUNT
L33	Main	10000
L35	Bazar	15000
L49	Market	9000
L98	Main	65000



Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

$\{t \mid t \in \text{loan} \wedge t[\text{amount}] \geq 10000\}$

OUTPUT:

LOAN NUMBER	BRANCH NAME	AMOUNT
L33	Main	10000
L35	Bazar	15000
L98	Main	65000

4.9. DOMAIN RELATIONAL CALCULUS

Domain Relational Calculus is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it. In Domain Relational Calculus, a query is expressed as,

$$\{ \langle X_1, X_2, X_3, \dots, X_n \rangle \mid P(X_1, X_2, X_3, \dots, X_n) \}$$

where, $\langle X_1, X_2, X_3, \dots, X_n \rangle$ represents resulting domains variables and $P(X_1, X_2, X_3, \dots, X_n)$ represents the condition or formula equivalent to the Predicate calculus.

Predicate Calculus Formula:

1. Set of all comparison operators
2. Set of connectives like and, or, not
3. Set of quantifiers

Example:



Table-1: Customer

CUSTOMER NAME	STREET	CITY
Ram	Gandhi Street	Vellore
Vijay	New Street	Chennai
Chitra	Gandhi Street	Chennai
Ravi	Bazar Street	Bangalore

Table-2: Loan

LOAN NUMBER	BRANCH NAME	AMOUNT
L33	Main	10000
L35	Bazar	15000
L49	Market	9000
L98	Main	65000

Table-3: Borrower

CUSTOMER NAME	LOAN NUMBER
Ramu	L01
Divya	L08
Sowmya	L03

Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

$\{ \langle l, b, a \rangle \mid \langle l, b, a \rangle \in \text{loan} \wedge (a > 100) \}$



OUTPUT

LOAN NUMBER	BRANCH NAME
L01	Main
L03	Main
L10	Sub

REVIEW QUESTIONS

1. Discuss the various types of *inner join* operations. Why is theta join required?
2. What role does the concept of *foreign key* play when specifying the most common types of meaningful join operations?
3. What is a join? Discuss different types of joins.
4. In what sense does relational calculus differ from relational algebra, and in what sense are they similar?
5. Why are tuples in a relation not ordered?
6. Why are duplicate tuples not allowed in a relation?
7. How does tuple relational calculus differ from domain relational calculus?
8. Discuss the meanings of the existential quantifier (\exists) and the universal quantifier (\forall).
9. Define the following terms with respect to the tuple calculus: *tuple variable*, *range relation*, *atom*, *formula*, and *expression*.
10. Define the following terms with respect to the domain calculus: *domain variable*, *range relation*, *atom*, *formula*, and *expression*.
11. What is meant by a *safe expression* in relational calculus?
12. When a query language is called relationally complete?



AIRPORT

<u>Airport_code</u>	Name	City	State
---------------------	------	------	-------

Figure 2:

FLIGHT

<u>Flight_number</u>	Airline	Weekdays
----------------------	---------	----------

FLIGHT_LEG

<u>Flight_number</u>	<u>Leg_number</u>	<u>Departure_airport_code</u>	<u>Scheduled_departure_time</u>
		<u>Arrival_airport_code</u>	<u>Scheduled_arrival_time</u>

LEG_INSTANCE

<u>Flight_number</u>	<u>Leg_number</u>	<u>Date</u>	<u>Number_of_available_seats</u>	<u>Airplane_id</u>	
		<u>Departure_airport_code</u>	<u>Departure_time</u>	<u>Arrival_airport_code</u>	<u>Arrival_time</u>

FARE

<u>Flight_number</u>	<u>Fare_code</u>	Amount	Restrictions
----------------------	------------------	--------	--------------

AIRPLANE_TYPE

<u>Airplane_type_name</u>	Max_seats	Company
---------------------------	-----------	---------

CAN_LAND

<u>Airplane_type_name</u>	<u>Airport_code</u>
---------------------------	---------------------

AIRPLANE

<u>Airplane_id</u>	Total_number_of_seats	Airplane_type
--------------------	-----------------------	---------------

SEAT_RESERVATION

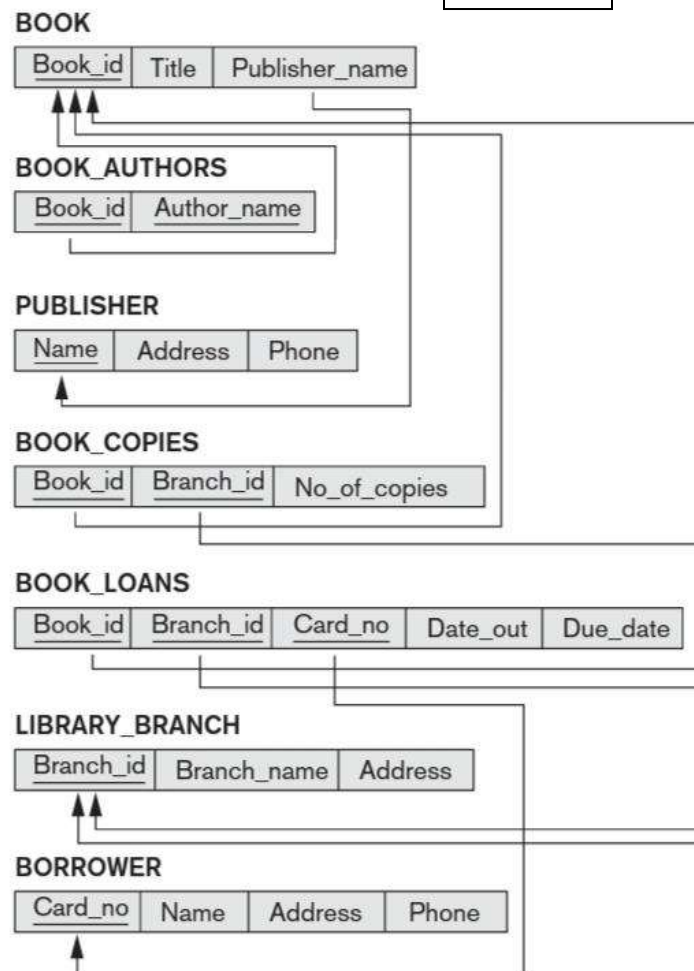
<u>Flight_number</u>	<u>Leg_number</u>	<u>Date</u>	<u>Seat_number</u>	Customer_name	Customer_phone
----------------------	-------------------	-------------	--------------------	---------------	----------------

13. Consider the AIRLINE relational database schema shown in Figure 2. Specify the following queries in relational algebra:
- For each flight, list the flight number, the departure airport for the first leg of the flight, and the arrival airport for the last leg of the flight.



- b. List the flight numbers and weekdays of all flights or flight legs that depart from Houston Intercontinental Airport (airport code 'IAH') and arrive in Los Angeles International Airport (airport code 'LAX').
- c. List the flight number, departure airport code, scheduled departure time, arrival airport code, scheduled arrival time, and weekdays of all flights or flight legs that depart from some airport in the city of Houston and arrive at some airport in the city of Los Angeles.
- d. List all fare information for flight number 'CO197'.
- b. Retrieve the number of available seats for flight number 'CO197' on '2009-10-09'.

Figure 3:





14. Consider the LIBRARY relational database schema shown in Figure 3. which is used to keep track of books, borrowers, and book loans. Referential integrity constraints are shown as directed arcs in Figure 3. Write down relational expressions for the following queries:
- How many copies of the book titled *The Lost Tribe* are owned by the library branch whose name is 'Sharpstown'?
 - How many copies of the book titled *The Lost Tribe* are owned by each library branch?
 - Retrieve the names of all borrowers who do not have any books checked out.
 - For each book that is loaned out from the Sharpstown branch and whose Due_date is today, retrieve the book title, the borrower's name, and the borrower's address.
 - For each library branch, retrieve the branch name and the total number of books loaned out from that branch. A relational database schema for a LIBRARY database.
 - Retrieve the names, addresses, and number of books checked out for all borrowers who have more than five books checked out.
 - For each book authored (or coauthored) by Stephen King, retrieve the title and the number of copies owned by the library branch whose name is Central.
15. Suppose that each of the following Update operations is applied directly to the database state shown in Figure 3. Discuss *all* integrity constraints violated by each operation, if any, and the different ways of enforcing these constraints.
- Insert <'Robert', 'F', 'Scott', '943775543', '1972-06-21', '2365 Newcastle Rd, Bellaire, TX', M, 58000, '888665555', 1> into EMPLOYEE.
 - Insert <'ProductA', 4, 'Bellaire', 2> into PROJECT.
 - Insert <'Production', 4, '943775543', '2007-10-01'> into DEPARTMENT.
 - Insert <'677678989', NULL, '40.0'> into WORKS_ON.
 - Insert <'453453453', 'John', 'M', '1990-12-12', 'spouse'> into DEPENDENT.
 - Delete the WORKS_ON tuples with Essn = '333445555'.
 - Delete the EMPLOYEE tuple with Ssn = '987654321'.
 - Delete the PROJECT tuple with Pname = 'ProductX'.



- i. Modify the Mgr_ssn and Mgr_start_date of the DEPARTMENT tuple with Dnumber = 5 to '123456789' and '2007-10-01', respectively.
 - j. Modify the Super_ssn attribute of the EMPLOYEE tuple with Ssn = '999887777' to '943775543'.
 - k. Modify the Hours attribute of the WORKS_ON tuple with Essn = '999887777' and Pno = 10 to '5.0'.
16. Consider the AIRLINE relational database schema shown in Figure 2, which describes a database for airline flight information. Each FLIGHT is identified by a Flight_number, and consists of one or more FLIGHT_LEGs with Leg_numbers 1, 2, 3, and so on. Each FLIGHT_LEG has scheduled arrival and departure times, airports, and one or more LEG_INSTANCES—one for each Date on which the flight travels. FAREs are kept for each FLIGHT.
- For each FLIGHT_LEG instance, SEAT_RESERVATIONs are kept, as are the AIRPLANE used on the leg and the actual arrival and departure times and airports. An AIRPLANE is identified by an Airplane_id and is of a particular AIRPLANE_TYPE. CAN_LAND relates AIRPLANE_TYPEs to the AIRPORTs at which they can land.
- An AIRPORT is identified by an Airport_code. Consider an update for the AIRLINE database to enter a reservation on a particular flight or flight leg on a given date.
- a. Give the operations for this update.
 - b. What types of constraints would you expect to check?
 - c. Which of these constraints are key, entity integrity, and referential integrity constraints, and which are not?
 - d. Specify all the referential integrity constraints that hold on the schema shown in Figure 2.
17. Consider the relation CLASS(Course#, Univ_Section#, Instructor_name, Semester, Building_code, Room#, Time_period, Weekdays, Credit_hours). This represents classes taught in a university, with unique Univ_section#s. Identify what you think should be various candidate keys, and write in your own words the conditions or assumptions under which each candidate key would be valid.



18. Consider the following six relations for an order-processing database application in a company:

CUSTOMER(Cust#, Cname, City)

ORDER(Order#, Odate, Cust#, Ord_amt)

ORDER_ITEM(Order#, Item#, Qty)

ITEM(Item#, Unit_price)

SHIPMENT(Order#, Warehouse#, Ship_date)

WAREHOUSE(Warehouse#, City)

Here, Ord_amt refers to total dollar amount of an order; Odate is the date the order was placed; and Ship_date is the date an order (or part of an order) is shipped from the warehouse. Assume that an order can be shipped from several warehouses. Specify the foreign keys for this schema, stating any assumptions you make. What other constraints can you think of for this database?

19. Consider the following relations for a database that keeps track of business trips of salespersons in a sales office:

SALESPERSON(Ssn, Name, Start_year, Dept_no)

TRIP(Ssn, From_city, To_city, Departure_date, Return_date, Trip_id)

EXPENSE(Trip_id, Account#, Amount)

A trip can be charged to one or more accounts. Specify the foreign keys for this schema, stating any assumptions you make.

20. Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

STUDENT(Ssn, Name, Major, Bdate)

COURSE(Course#, Cname, Dept)

ENROLL(Ssn, Course#, Quarter, Grade)

BOOK_ADOPTION(Course#, Quarter, Book_isbn)



TEXT(Book_isbn, Book_title, Publisher, Author)

Specify the foreign keys for this schema, stating any assumptions you make.

21. Consider the following relations for a database that keeps track of automobile sales in a car dealership (OPTION refers to some optional equipment installed on an automobile):

CAR(Serial_no, Model, Manufacturer, Price)

OPTION(Serial_no, Option_name, Price)

SALE(Salesperson_id, Serial_no, Date, Sale_price)

SALESPERSON(Salesperson_id, Name, Phone)

First, specify the foreign keys for this schema, stating any assumptions you make. Next, populate the relations with a few sample tuples, and then give an example of an insertion in the SALE and SALESPERSON relations that *violates* the referential integrity constraints and of another insertion that does not.

22. Database design often involves decisions about the storage of attributes. For example, a Social Security number can be stored as one attribute or split into three attributes (one for each of the three hyphen-delineated groups of numbers in a Social Security number—XXX-XX-XXXX). However, Social Security numbers are usually represented as just one attribute. The decision is based on how the database will be used. This exercise asks you to think about specific situations where dividing the SSN is useful.

23. Consider a STUDENT relation in a UNIVERSITY database with the following attributes (Name, Ssn, Local_phone, Address, Cell_phone, Age, Gpa). Note that the cell phone may be from a different city and state (or province) from the local phone. A possible tuple of the relation is shown below:

Name Ssn Local_phone Address Cell_phone Age Gpa

George Shaw 123-45-6789 555-1234 123 Main St., 555-4321 19 3.75

William Edwards Anytown, CA 94539



- a. Identify the critical missing information from the Local_phone and Cell_phone attributes. (*Hint: How do you call someone who lives in a different state or province?*)
 - b. Would you store this additional information in the Local_phone and Cell_phone attributes or add new attributes to the schema for STUDENT?
 - c. Consider the Name attribute. What are the advantages and disadvantages of splitting this field from one attribute into three attributes (first name, middle name, and last name)?
 - d. What general guideline would you recommend for deciding when to store information in a single attribute and when to split the information?
 - e. Suppose the student can have between 0 and 5 phones. Suggest two different designs that allow this type of information.
24. Recent changes in privacy laws have disallowed organizations from using Social Security numbers to identify individuals unless certain restrictions are satisfied. As a result, most U.S. universities cannot use SSNs as primary keys (except for financial data). In practice, Student_id, a unique identifier assigned to every student, is likely to be used as the primary key rather than SSN since Student_id can be used throughout the system.
- a. Some database designers are reluctant to use generated keys (also known as *surrogate keys*) for primary keys (such as Student_id) because they are artificial. Can you propose any natural choices of keys that can be used to identify the student record in a UNIVERSITY database?
 - b. Suppose that you are able to guarantee uniqueness of a natural key that includes last name. Are you guaranteed that the last name will not change during the lifetime of the database? If last name can change, what solutions can you propose for creating a primary key that still includes last name but remains unique?
 - c. What are the advantages and disadvantages of using generated (surrogate) keys?



OBJECTIVE TYPE QUESTIONS

1. Which of the following represents a relationship among a set of values.

- a) A Row
- b) A Table
- c) A Field
- d) A Column

2. Column header is refer as

- a) Table
- b) Relation
- c) Attributes
- d) Domain

3. Which algebra is widely used in DBMS?

- a) Relational algebra
- b) Arithmetic algebra
- c) Both
- d) None

4. A Relation is a

- a) Subset of a Cartesian product of a list of attributes
- b) Subset of a Cartesian product of a list of domains
- c) Subset of a Cartesian product of a list of tuple
- d) Subset of a Cartesian product of a list of relations

5. In mathematical term Table is referred as

- a) Relation
- b) Attribute
- c) Tuple
- d) Domain

6. In mathematical term Row is referred as

- a) Relation
- b) Attribute
- c) Tuple
- d) Domain

7. _____ Allow us to identify uniquely a tuple in the relation.

- a) Super key
- b) Domain
- c) Attribute
- d) Schema

8. Which of the following is Relation-algebra Operation

- a) Select
- b) Union
- c) Rename
- d) All of the above

9. Set of premitted values of each attribute is called

- a) Domain
- b) Tuple
- c) Relation
- d) Schema



10. In tuple relational calculus $P1 \oplus P2$ is equivalent to

- a) $\neg P1 \cup P2$
- b) $P1 \cup P2$
- c) $P1 \cap P2$
- d) $P1 \cap \neg P2$

11. Relational Algebra is

- a) Data Definition Language .
- b) Meta Language
- c) Procedural query Language
- d) None of the above

12. _____ produces the relation that has attributes of R1 and R2

- a) Cartesian product
- b) Difference
- c) Intersection
- d) Product

13. Cartesian product in relational algebra is

- a) A Unary operator.
- b) A Binary operator.
- c) A Ternary operator.
- d) Not defined.

14. DML is provided for

- a) Description of logical structure of database.
- b) Addition of new structures in the database system.
- c) Manipulation & processing of database.
- d) Definition of physical structure of database system.

15. Which of the following Relational Algebra operations require that both tables (or virtual tables) involved have the exact same attributes/data types?

- a) Join, Projection, Restriction
- b) Multiplication and Division
- c) Union, Intersection, Minus
- d) Minus, Multiplication, Intersection

16. Logical design of database is called

- a) Database Instance
- b) Database Snapshot
- c) Database Schema
- d) All of the above

17 The result of the UNION operation between R1 and R2 is a relation that includes

- a) All the tuples of R1
- b) All the tuples of R2
- c) All the tuples of R1 and R2
- d) All the tuples of R1 and R2 which have common columns

18. Which of the following is not Unary operation?

- a) Select
- b) Project
- c) Rename
- d) Union



19. Which of the following is not binary operation?

- a) Union
- b) Project
- c) Set Difference
- d) Cartesian Product

20. Which of the following is not Outer join?

- a) Left outer join
- b) Right outer join
- c) Full outer join
- d) All of the above

21. A file manipulation command that extracts some of the records from a file is called

- a) SELECT
- b) PROJECT
- c) JOIN
- d) PRODUCT

22. What does the following SQL statement do?

Select * From Customer Where Cust_Type = "Best";

- a) Selects all the fields from the Customer table for each row with a customer labeled "best"
- b) Selects the "*" field from the Customer table for each row with a customer labeled "best"
- c) Selects fields with a "*" in them from the Customer table
- d) Selects all the fields from the Customer table for each row with a customer labeled "*"

23. In an SQL statement, which of the following parts states the conditions for row selection?

- a) Select
- b) From
- c) Order By
- d) Where

24. Cross Product is a:

- a) Unary Operator
- b) Ternary Operator
- c) Binary Operator
- d) Not an operator

25. 'AS' clause is used in SQL for

- a) Selection operation.
- b) (B) Rename operation.
- c) (C) Join operation.
- d) (D) Projection operation

26. Which of the following is correct:

- a) SQL query automatically eliminates duplicates.
- b) (B)SQL permits attribute names to be repeated in the same relation.
- c) SQL query will not work if there are no indexes on the relations
- d) (D)None of these

27. Which of the following is a legal expression in SQL?

- a) SELECT NULL FROM EMPLOYEE;
- b) SELECT NAME FROM EMPLOYEE;
- c) SELECT NAME FROM EMPLOYEE WHERE SALARY = NULL;
- d) None of the above



28. A set of possible data values is called

- a) Attribute.
- b) Degree.
- c) Tuple.
- d) Domain

29. Which of the operations constitute a basic set of operations for manipulating relational data?

- a) Predicate calculus
- b) Relational calculus
- c) Relational algebra
- d) None of the above

KEYS

1-a, 2-c, 3-a, 4-b, 5-a, 6-c, 7-a, 8-d, 9-a, 10-c, 11-c, 12-a, 13-b, 14-c, 15-c,
16-c, 17-d, 18-d, 19-b, 20-d, 21-a, 22-a, 23-d, 24-c, 25-b, 26-d, 27-b, 28-d, 29-c.



UNIT – III

CHAPTER 5: NORMALIZATION

5.1. INTRODUCTION

The process of normalization was first developed by E. F. Codd. Database designed based on E-R model may have some amount of inconsistency, uncertainty and redundancy.

Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process:

- **Eliminating redundant data** and
- **Ensuring data dependencies make sense** (only storing related data in a table).

5.2. What is Normalization?

Normalization is defined as a step by step process of decomposing a complex relation into simple and stable relations.

Advantages of Normalization

- Faster search performance
- Data integrity is easily maintained within the database.
- Reduces data redundancy in a database(So Decreased storage space)
- Remove Insert, Delete, Update anomalies during database activates
- Security is easier to maintain or manage.

Disadvantages of Normalization

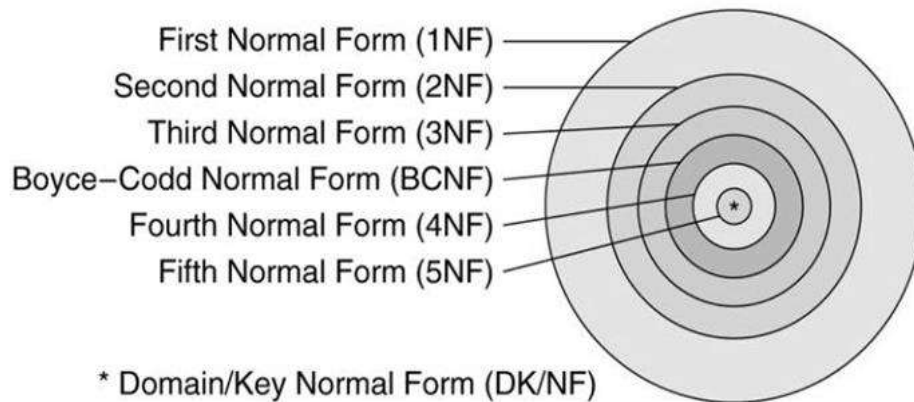
- **Performance:** all the joins required to merge data slow processing & place additional stress on your hardware.
- **Complex queries:** developers have to code complex queries in order to merge data from different tables.
- It is very time consuming and difficult process in normalizing relations of higher degree.



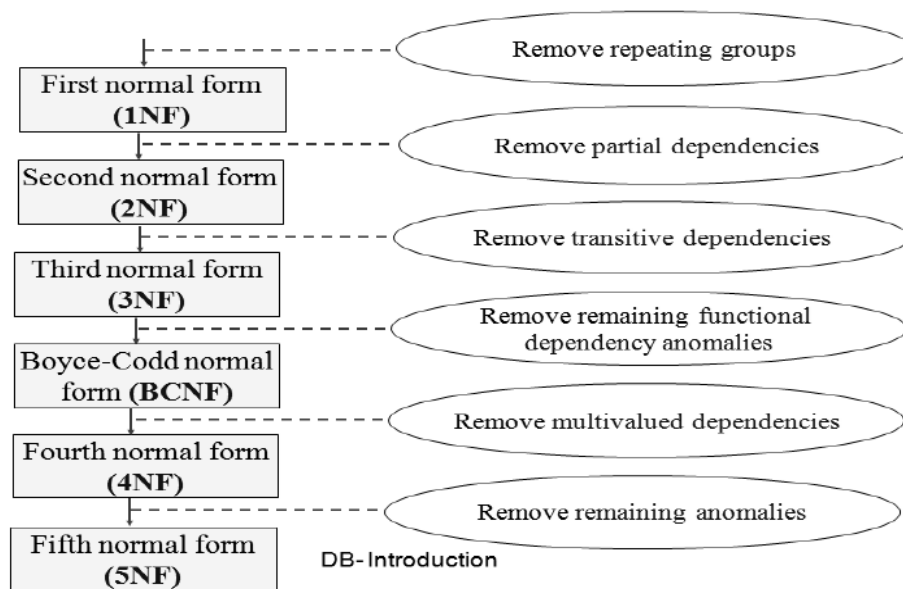
The Normal Forms

The different stages of Normalization are known as “**Normal Form**”. (It referred to as first normal form or 1 NF, 2 NF, 3 NF, BCNF, 4 NF and 5 NF (PJNF)). Normalization process is based on functional dependency.

Figure Relationship of Normal Forms



Stages of Normalisation





5.3. FUNCTIONAL DEPENDENCY

A functional dependency (FD) is a constraint between two sets of attributes from the database. A functional dependency denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of relation R .

It specifies a constraint on the possible tuples that can form a relation instance r of R . The constraint states that for any two tuples t_1 and t_2 in r such that $t_1[X] = t_2[X]$. This means that values of the X component of a tuple uniquely or functionally determine the values of the Y component.

Functional dependency is a term derived from mathematical theory, which states that for every element in the attribute. For example, given the value of item code, there is only one value of item name for it. Thus item name is functionally dependent on item code. This is shown as:

Item code \rightarrow item name

Functional dependency may also be based on a composite attribute. For example, if we write

{X, Y} \rightarrow Z

It means that Z is functionally dependent on composite attribute X and Y .

A functional dependency is a property of the meaning or semantics of the attributes in a relation schema R . The functional dependencies can denote as,

Name \rightarrow {Ph.No., Major}

Course \rightarrow Prof

{Name, Course} \rightarrow Grade

Where, in the first relation, the attributes Ph.No and Major are functionally dependent on the prime attribute Name. Alternatively, we can say that the prime attribute Name determined the non prime attributes Ph.No and Major. Similarly we can now explain the other two relational schemas also.



TYPES OF FUNCTIONAL DEPENDENCY

1. Full functional dependency
2. Partial functional dependency
3. Transitive functional dependency
4. Multi-Valued Dependency
5. Join Dependency

5.3.1. FULL FUNCTIONAL DEPENDENCY

When all non-key attributes are dependent on the key attribute, it is called full functional dependency. For example, consider the relation schema of an employee,

The term full functional dependency (FFD) is used to indicate the minimum set of attributes in a determinant of a functional dependency (FD). In other words, the set of attributes X will be fully functionally dependent on the set of attributes Y if the following conditions are satisfied:

- X is functionally dependent on Y and
- X is not functionally dependent on any subset of Y.
 - In relation ASSIGN
FD: {EMP-ID, PROJECT, PROJECT-BUDGET} → {YRS-SPENT-BY-EMP-ON-PROJECT}
 - The values of EMP-ID, PROJECT and PROJECT-BUDGET determine a unique value of YRS-SPENT-BY-EMP-ON-PROJECT.
 - However, it is not a full functional dependency because neither the EMP-ID → YRS-SPENT-BY-EMP-ON-PROJECT nor the PROJECT → YRS-SPENT-BY-EMP-ON-PROJECT holds true.
 - In fact, it is sufficient to know only the value of a subset of {EMP-ID, PROJECT, PROJECT-BUDGET}, namely, {EMP-ID, PROJECT}, to determine the YRS-SPENT-BY-EMP-ON-PROJECT.
 - Thus, the correct full functional dependency (FFD) can be written as:
FD: {EMP-ID, PROJECT} → {YRS-SPENT-BY-EMP-ON-PROJECT}



Relation R1 : BUDGET

PROJECT	PROJECT - BUDGET
P1	INR 100 CR
P2	INR 150 CR
P3	INR 200 CR
P4	INR 100 CR
P5	INR 150 CR
P6	INR 300 CR

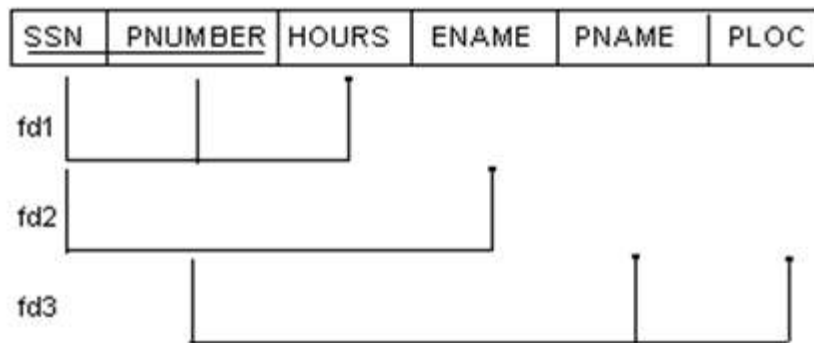
Relation R2: ASSIGN

EMP – NO	PROJECT	YRS – SPENT
		BY EMP-ON-PROJECT
106519	P1	5
112233	P3	2
106519	P2	5
123243	P4	10
106519	P3	3
111222	P1	4



5.3.2. PARTIAL DEPENDENCY

A functional dependency $X \rightarrow Y$ is a partial dependency if some attributes $A \rightarrow X$ can be removed from X and the dependency still holds. It means that all non-key attributes are not dependent on the key attribute. There is a partial dependency of non-key attributes either on the key attribute or on the non-key attribute.

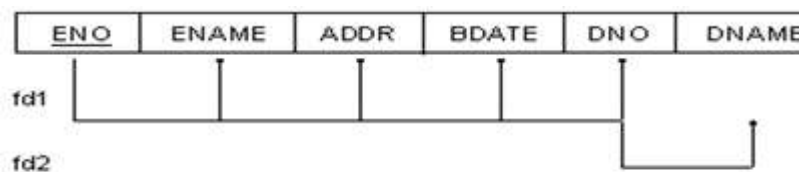


A relation schema of Emp-Project relation

In the above figure, the dependency $\{SSN, PNUMBER\} \rightarrow ENAME$ is partial because $SSN \rightarrow ENAME$ holds

5.3.3. TRANSITIVE DEPENDENCY

A functional dependency $X \rightarrow Y$ in a relation R is a transitive dependency if there is a set of attributes Z that is not a subset of any key of R and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold. In general sense we can say that if A, B, C are three attributes in a table, and if A is related to B and B is related to C , then A is indirectly related to C .



A relation schema of Emp-Dept relation



In the above figure, DNO is dependent on key attribute ENO and DNAME is dependent on DNO which we can denote as,

ENO \rightarrow {ENAME, ADDR, BDATE, DNO}

DNO \rightarrow DNAME

From the above dependencies we can say that DNAME is indirectly related to key attribute ENO. So, DNAME is transitively dependent on ENO.

5.3.4. MULTI-VALUED DEPENDENCY

Let R be a relational variable and A, B and C are subsets of the attributes of R. Then we say that B is multi-dependent on A. Symbolically, it can be denoted as,

A \twoheadrightarrow B and read as A multi-determines B

Multi-valued dependencies are a generalization of functional dependencies, in the sense that every functional dependency is an multi-valued dependency. But the converse is not true

5.3.5. JOIN DEPENDENCY

Join dependency is a constraint, similar to a Functional dependency or a Multi-valued dependency. It is satisfies if and only if the relation concerned is the join of certain number of projections. The definition is if R be a relational variable, and A, B,, Z are subsets of attributes of R, then R satisfies the Join Dependency that is denoted a,

* {A, B,, Z} and read as star A, B,, Z

5.4. NORMAL FORMS

5.4.1. FIRST NORMAL FORM (1 NF)

DEFINITION

A relation is in 1NF if and only if all underlying domains contain scalar values only.

PURPOSE

The purpose of this Normal form is to ensure that a table does not contain any multi-part or multi-valued fields, and that each field holds only a single value for any given record.



- The objective of first normal form is that the table should contain no repeating groups of data.
- Data is divided into logical units called entities or tables (Note: when each entity has been designed, a primary key is assigned to it.)
- All attributes (column) in the entity (table) must be single valued.
- Repeating or multi valued attributes are moved into a separate entity (table) & a relationship is established between the two tables or entities.

Example:

EMPLOYEE (UN-NORMALIZED DATA)

Emp_No	Name	Dept_No	Dept_Name	Skills
1	Ravi	201	R & D	C, PERL, JAVA
2	Meena	224	IT	LINUX, MAC
3	Kala	201	R & D	DB2, ORACLE, JAVA

The First normal form says that each field in a table must contain only a single type of data and each piece of data must be stored in only one place. This results in isolating of repeating groups within an entity. But in the above example SKILLS fields have redundancy.

Problem: Employee relation schema with lack of atomicity in skills (attribute). Suppose you want to delete the dept_no as '201' then you lose two data. (Because of redundancy data) It may call Insert, Delete, Update anomalies during database activates

Solution: Make a separate table for each set of attributes with a primary key

EMPLOYEE (1 NF)

Emp_No	Name	Dept_No	Dept_Name	Skills
1	Ravi	201	R & D	C



1	Ravi	201	R & D	PERL
1	Ravi	201	R & D	JAVA
2	Meena	224	IT	LINUX
2	Meena	224	IT	MAC
3	Kala	201	R & D	DB2
3	Kala	201	R & D	ORACLE
3	Kala	201	R & D	JAVA

Although the table is far from perfect, it is now in First Normal Form and is ready to be tested against Second Normal form

ADVANTAGE

- Easier to query/sort the data:
- More scalable
- Each row can be identified for updating

DISADVANTAGE

This solution has the disadvantage of introducing redundancy in relation. So we move on to 2NF.

5.4.2. SECOND NORMAL FORM (2 NF)

DEFINITION

A relation is in 2 NF if and only if it is in 1 NF and every non- key attribute is fully dependent on the primary key.

- The objective of second normal form is that every field in a table should relate to the primary key field in its entity.



- It means that data that is only partly dependent on the primary key is stored into another table
- To bring table into a second normal form, it should be first in first normal form.

Example:

EMPLOYEE (1 NF)

Emp_No	Name	Dept_No	Dept_Name	Skills
1	Ravi	201	R & D	C
1	Ravi	201	R & D	PERL
1	Ravi	201	R & D	JAVA
2	Meena	224	IT	LINUX
2	Meena	224	IT	MAC
3	Kala	201	R & D	DB2
3	Kala	201	R & D	ORACLE
3	Kala	201	R & D	JAVA

Name, dept_no, and dept_name are functionally dependent on emp_no.(emp_no -> name, dept_no, dept_name)

SKILLS (2NF)

Emp_No	Skills
1	C
1	PERL
1	JAVA
2	LINUX
2	MAC
3	DB2
3	ORACLE
3	JAVA

Problem: Skills(attribute) is not functionally dependent on emp_no since it is not unique to each emp_no.



EMPLOYEE (2 NF)

Emp_No	Name	Dept_No	Dept_Name
1	Ravi	201	R & D
2	Meena	224	IT
3	Kala	201	R & D

Solution: Make a separate table like emp _ details (emp_no, name, dept_no, dept_name) and skill_ details (emp_no, skills) and to set a primary key for emp_no attribute. The resulting tables must be related to each other by use of foreign key.

Now that we have removed the duplicative data from the employee table, and it have to be moved to a separate table. We've satisfied the first rule of second normal form. We still need to use a foreign key to tie the two tables together. The result table looks like.

Employee (2NF)

Emp_no	Name	Dept_no	Dept_name	skills

Benefits:

- Decrease storage efficiency
- Less data repetition

Disadvantage:

This solution has the disadvantage of introducing redundancy in skills relation. So we move on to 3NF.

5.4.3. THIRD NORMAL FORM (3 NF)

Definition:

A relation is in 3 NF if and only if it is in 2 NF and every non- key attribute is non-transitively dependent (mutual dependent) on the primary key.

- The objective of third normal form is to remove field /data in a table that is not dependent on the primary key. It means that, any non-key (not fully) field in a table must relate to the primary key of the table& not to any other field.



- The entity is in second normal form and non-key attributes cannot depend on another non-key attributes
- All non-key attributes should depend directly on the whole primary key and not on each other.

Example:

EMPLOYEE (2 NF)

Emp_No	Name	Dept_No	Dept_Name
1	Ravi	201	R & D
2	Meena	224	IT
3	Kala	201	R & D

Dept_no and dept_name are functionally dependent on emp_no however, department can be considered a separate entity.

EMPLOYEE (3 NF)

Emp_No	Name	Dept_No
1	Ravi	201
2	Meena	224
3	Kala	201

DEPARTMENT (3NF)

Emp_No	Dept_No	Dept_Name
1	201	R & D
2	224	IT

Here #emp_no, #dept_no are primary key attribute.

Benefits: No extraneous (coming from the outside) data.



5.4.4. Boyce- Code NORMAL FORM (BCNF)

Definition: A relation is in BCNF if and only if the only determinants are candidate keys (more than one unique key).

- It is based on FD that takes into account all candidate keys in a relation.
- A relation is said to be in BCNF if and only if every determinant is a candidate key.
- A determinant is an attribute or a group of attributes on which some other attribute is fully functionally determinant
- To test whether a relation is in BCNF, we identify all the determinants and make sure that they are candidate keys.
 - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
 - α is a superkey for R

Example:

$R = (A, B, C)$

$F = \{A \rightarrow B, B \rightarrow C\}$

Key = $\{A\}$

Here R is not in BCNF. Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$, R_1 and R_2 in BCNF, Lossless-join decomposition.

Consider a scenario of a large development organization, where the projects are organized in project groups, each with a team leader acting as a liaison between the overall project and a group of developers in a matrix organization. Assume we have the following situation: Each Project can have many Developers.

- Each Developer can have many Projects.
- For a given Project, each Developer only works for one Lead Developer.
- Each Lead Developer only works on one Project.
- A given Project can have many Lead Developers.

In this case, we could theoretically design a table in two different ways:



Project No	Developer	Lead Developer
20020123	John Doe	Elmer Fudd
20020123	Jane Doe	Sylvester
20020123	Jimbo	Elmer Fudd
20020124	John Doe	Ms. Depesto

Case 1: Project Number and Developer as a Candidate Key can be used to determine the Lead Developer. In this case, the Lead Developer depends on both attributes of the key, and the table is 3NF if we consider that our Primary Key.

Lead Developer	Developer	Project No
Elmer Fudd	John Doe	20020123
Sylvester	Jane Doe	20020123
Elmer Fudd	Jimbo	20020123
Ms. Depesto	John Doe	20020124

Case 2: Lead Developer and Developer is another Candidate Key, but in this case, the Project Number is determined by the Lead Developer alone.

Thus it would not be 3NF if we consider that our Primary Key. In reality, these three data items contain more than one relation (Project - Lead Developer and Lead Developer - Developer). To normalize to BCNF, we would remove the second relation and represent it in a second table. (This also illustrates why a table is formally named a relation.)

ProjectNo	Lead Developer
20020123	Elmer Fudd
20020123	Sylvester
20020123	Elmer Fudd
20020124	Ms. Depesto



5.4.5. FORTH NORMAL FORM (4NF)

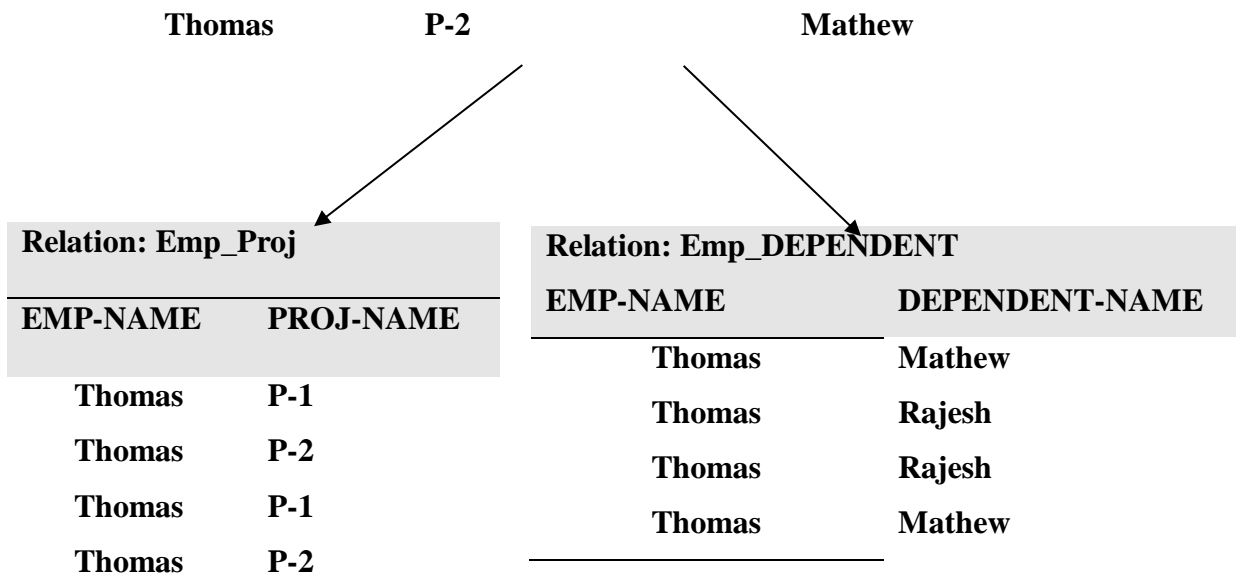
Definition: A table is in Boyce-Code normal form (BCNF) and there are no multi-valued dependencies.

- The fourth normal form (4NF) is concerned with dependencies between the elements of compound keys composed of three or more attributes
- The 4NF eliminates the problems of 3NF. 4NF is violated when a relation has undesirable MVDs and hence can be used to identify and decompose such relations

Example 1

- Relation EMPLOYEE, as shown in Fig.
- A tuple in this relation represents the fact that an employee (EMP-NAME) works on the project (PROJ-NAME) and has a dependent (DEPENDENT-NAME).
- This relation is not in 4NF because in the non-trivial MVDs $EMP-NAME \twoheadrightarrow PROJ-NAME$
- And $EMP-NAME \twoheadrightarrow DEPENDENT-NAME$, $EMP-NAME$ is not a super key of EMPLOYEE.
- Now the relation EMPLOYEE is decomposed into EMP_PROJ and EMP_DEPENDENTS.
- Thus, both EMP_PROJ and EMP_DEPENDENT are in 4NF, because the MVDs $EMP-NAME \twoheadrightarrow PROJ-NAME$ in EMP_PROJ
- And $EMP-NAME \twoheadrightarrow DEPENDENT-NAME$ in EMP_DEPENDENTS are trivial MVDs.
- No other non-trivial MVDs hold in either EMP_PROJ or EMP_DEPENDENTS. No FDs hold in these relation schemas either.

Relation EMPLOYEE		
EMP-NAME	PROJ-NAME	DEPENDENT-NAME
Thomas	P-1	Mathew
Thomas	P-2	Rajesh
Thomas	P-1	Rajesh



PROBLEMS WITH MVDS AND 4NF

- FDs, MVDs and 4NF are not sufficient to identify all data redundancies.
- Let us consider a relation PERSONS_ON_JOB_SKILLS, as shown in Table.
- This relation stores information about people applying all their skills to the jobs to which they are assigned. But, they use particular or all skills only when the job needs that skill.

Relation: Persons_on_Job_Skills			
Person	Skill-Type	Job	
Thomas	Analyst	J-1	
Thomas	Analyst	J-2	
Thomas	DBA	J-2	
Thomas	DBA	J-3	
John	DBA	J-1	
Antony Paul Raj	Analyst	J-1	

- The relation PERSONS_ON_JOB_SKILLS of Table is in BCNF and 4NF.



- For example, person "Thomas" who possesses skills "Analyst" and "DBA" applies them to job J-2, as J-2 needs both these skills.
- The same person "Thomas" applies skill "Analyst" only to job J-1, as job J-1 needs only skill "Analyst" and not skill "DBA".
- Thus, if we delete <Thomas, DBA, J-2>, we must also delete <Thomas, Analyst, J-2>, because persons must apply all their skills to a job if that requires those skills.

5.4.6. JOIN DEPENDENCIES AND FIFTH NORMAL FORM (5NF)

The anomalies of MVDs are eliminated by join dependency (JD) and 5NF.

1. JOIN DEPENDENCIES (JD)

- A join dependency (JD) can be said to exist if the join of R1 and R2 over C is equal to relation R.
 - Where, R1 and R2 are the decompositions R1(A, B, C), and R2(C, D) of a given relations R(A, B, C, D).
- R1 and R2 is a lossless decomposition of R.
- In other words, *(A, B, C, D), (C, D) will be a join dependency of R if the join of the join's attributes is equal to relation R.
 - *(R1, R2, R3 ...) indicates that relations R1, R2, R3 and soon are a join dependency (JD) of R.
- Relation R to satisfy a JD *(R1, R2, ..., RN) is that
 - $R = R1 \cup R2 \cup \dots \cup RN$.
- Thus, whenever we decompose a relation R into R1 = XUY and R2 = (R - Y) based on an MVD $X \twoheadrightarrow Y$ that holds in relation R, the decomposition has lossless join property.
- Therefore, lossless-join dependency can be defined as a property of decomposition, which ensures that no spurious tuples are generated when relations are returned through a natural join operation.

Example 1

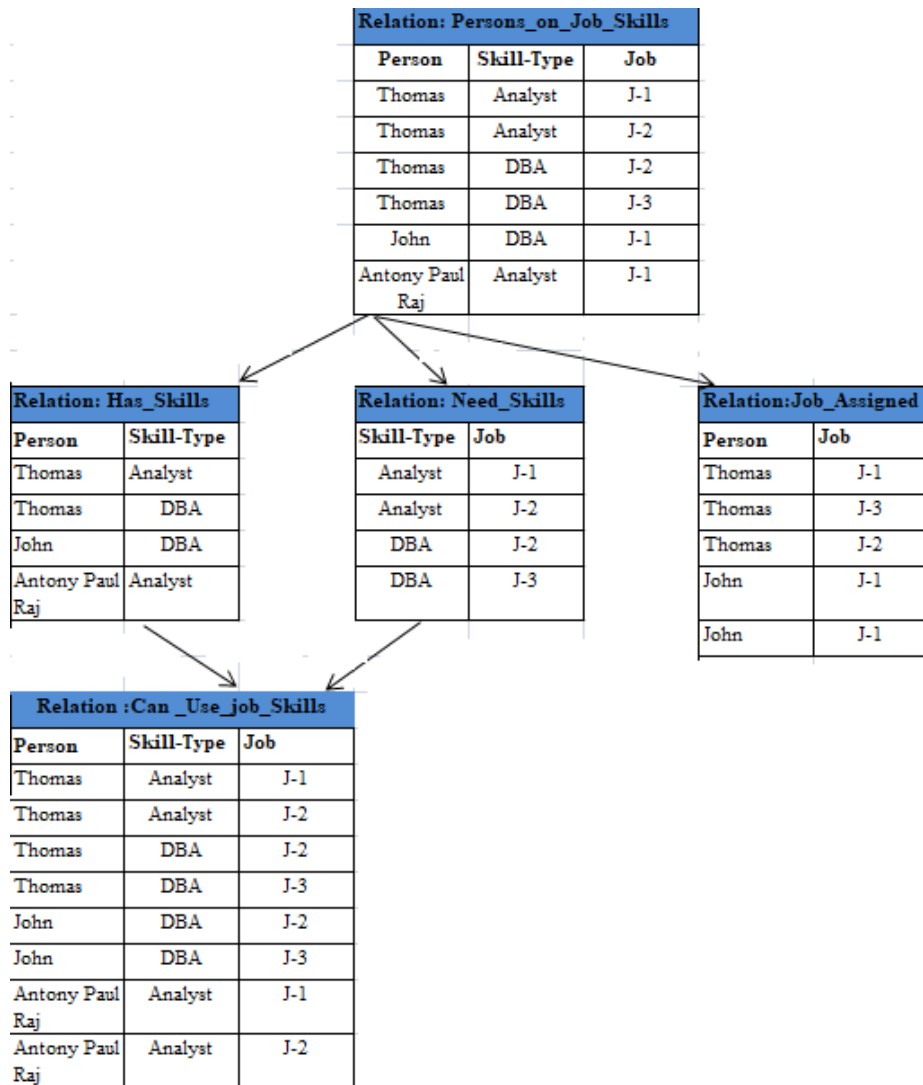
- Relation PERSONS_ON_JOB_SKILLS, as shown in above fig .



- This relation can be decomposed into three relations namely, HAS_SKILL, NEEDS_SKILL and ASSIGNED_TO_JOBS.
 - fig. the join dependencies of decomposed relations.
- if we join decomposed relations HAS_SKILL and NEEDS_SKILL, a relation CAN_USE_JOB_SKILL is obtained, as shown in above Fig.

This relation stores the data about persons who have skills applicable to a particular job. But, each person who has a skill required for a particular job need not be assigned to that job.

Thus, redundant tuples (rows) that show unnecessary SKILL-TYPE and JOB combinations are removed by joining with relation NEEDS_SKILL.



2. FIFTH NORMAL FORM (5NF)

Definition. A relation schema R is in **fifth normal form (5NF)** (or **project-join normal form (PJNF)**) with respect to a set F of functional, multivalued, and join dependencies if, for every nontrivial join dependency $JD(R_1, R_2, \dots, R_n)$ in F^+ (that is, implied by F),¹⁸ every R_i is a superkey of R .

- For every non-trivial join dependency $*(R_1 R_2 R_3)$ each decomposed relation R_i is a super key of the main relation R .
- 5NF is also called project-join normal form (PJNM).
- There are some relations, who cannot be decomposed into two or higher normal form relations by means of projections as discussed in 1NF, 2NF, 3NF and BCNF.



- Relations are decomposed into three or more relations, which can be reconstructed by means of a three-way or more join operation. This is called fifth normal form (5NF).
- The 5NF eliminates the problems of 4NF. 5NF allows for relations with join dependencies.
- Any relation that is in 5NF, is also in other normal forms namely 2NF, 3NF and 4NF.
- 5NF is mainly USED FROM theoretical point of view and not for practical database design

Example 1

- Let us consider the relation PERSONS_ON_JOB_SKILLS of Fig.
- The three relations are
 - HAS_SKILL (PERSON, SKILL-TYPE)
 - NEEDS_SKILL (SKILL-TYPE, JOB)
 - JOB_ASSIGNED (PERSON, JOB)
- Now by applying the definition of 5NF, the join dependency is given as:
 - $*((PERSON, SKILL-TYPE), (SKILL-TYPE, JOB), (PERSON, JOB))$
- The above statement is true because a join relation of these three relations is equal to the original relation PERSONS_ON_JOB_SKILLS.
- The consequence of these join dependencies is that the SKILL-TYPE, JOB or PERSON, is not relation key, and hence the relation is not in 5NF. Now suppose, the second tuple (row 2) is removed from relation PERSONS_ON_JOB_SKILLS, a new relation is created that no longer has any join dependencies. Thus the new relation will be in 5NF.

5.4.7. DOMAIN KEY NORMAL FORM (DKNF)

The idea behind DKNF is to specify the “ultimate normal form” that takes into account all possible types of dependencies and constraints.

Domain/Key Normal Form

Relation R is said to be in DKNF if and only if every constraint on R is a logical Consequence of the domain constraints and key constraints that apply to R.



Domain/Key Normal Form is a newer normal form and is similar to BCNF in that it is partially based on the enforcement of primary keys and candidate keys. So you already understand at least that much of this Normal Form. But it is also based upon the idea of Domain. Most texts on database design state that a Domain is merely a set of acceptable values from which a specific field can draw its own values. That's only partially true; a Domain is much more than. A Domain has two sides; a logical side and a physical side. The logical side deals with issues such as default value range of values, whether the value is required and whether the value can be Null. The physical side deals with issues such as data type, length, decimal places and allowable characters. Once you understand this idea, you can use this Normal Form

In order for a table to be in Domain/Key Normal Form it must fulfil these requirements.

1. Each field must be fully and properly defined.
2. Each field must represent a characteristic of the table's subject.
3. Each non – key field in the table must be functionally dependent upon the entire primary key.
4. Each table should represent only a single Subject

A table DKNF will be free of transitive dependencies, Multi – Valued dependencies and modification anomalies. In fact, a table in DKNF is automatically in Fifth Normal Form

Review Questions

1. Why do we need normalization?
2. Explain the functional dependency with multi-valued dependencies with example.
3. Explain 3NF with example and Compare BCNF and 3NF.
4. Explain 4NFs. How it is different from other normal forms?
5. What is 3NF?
6. What is functional dependency?
7. Explain the purpose of normalization and schema refinement.



8. Explain the role of minimal cover for set of FDs in 3rd normal form.
9. What is the difference between controlled and uncontrolled redundancy? Illustrate with examples.

OBJECTIVE TYPE QUESTIONS

1. A relation that has no partial dependencies is in which normal form

- | | |
|-----------|----------|
| a) First | c) Third |
| b) Second | d) BCNF |

2. A functional dependency between two or more non-key attributes is called

- | | |
|----------------------------------|----------------------------------|
| a) Transitive dependency | c) Functional dependency |
| b) Partial transitive dependency | d) Partial functional dependency |

3. If K is a foreign key in a relation R1, then

- a) Every tuple of R1 has a distinct value for K
- b) K cannot have a null value for tuples in R1
- c) K is a key for some other relation
- d) K is a Primary key for R1

4. Which of the following concept is applicable with respect to 2NF?

- a) Full functional dependency
- b) Partial dependency
- c) Transitive dependency
- d) Non-transitive dependency

5. If every non-key attribute is functionally dependent on the primary key, the relation will be in

- | | |
|-----------------------|-----------------------------|
| a) First Normal Form | c) Third Normal Form |
| b) Second Normal Form | d) Fourth Normal Form |

6. In DBMS FD stands for _____

- | | |
|--------------------|--------------------------|
| a) Facilitate data | c) Facilitate dependency |
| b) Functional data | d) Functional dependency |

7. Which of the following is based on Multi Valued Dependency?

- | | |
|-----------|-----------|
| a) First | c) Third |
| b) Second | d) Fourth |



8. Which of the following is correct?

- a) Function dependencies are not associated with relations; they are based on the semantics of information that we are dealing with
- b) If a relation has no redundant information its attributes must not have any function dependencies
- c) Functional dependencies may be determined if we are given several instances of a relation
- d) The FDs that hold for attributes of a relation need not be satisfied at all times

9. A functional dependency between two or more non-key attributes is called

- a) Partial functional dependency
- b) Partial non-key dependency
- c) Transitive dependency
- d) Partial transitive dependency

KEYS

1-b, 2-a, 3-c, 4-a, 5-c, 6-d, 7-d, 8-a, 9-c



UNIT – IV

CHAPTER 6: STRUCTURED QUERY LANGUAGE

6.1. INTRODUCTION

Structured Query Language (SQL) is a standard computer language for storing, manipulating and retrieving data in databases. SQL is used to communicate with a database. According to ANSI, it is the standard language for relational database management systems.

It is the special purpose domain specific language for querying the data in Relational Database Management System (RDBMS). An SQL developer must decide what type of data that will be stored inside each column when creating a table.

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

6.2. STRUCTURED QUERY LANGUAGE

Structured Query Language, commonly abbreviated to SQL and pronounced as “sequel”, is not a conventional computer programming language in the normal sense of the phrase. It allows users to access data in relational database management systems.

SQL is about data and results; each SQL statement returns a result, whether that result be a query, an update to a record or the creation of a database table. SQL is most often used to address a relational database, which is what some people refer to as a SQL database.

So in brief we can describe SQL as follows:

- SQL stands for Structured Query Language
- SQL allows you to access a database
- SQL can execute queries against a database

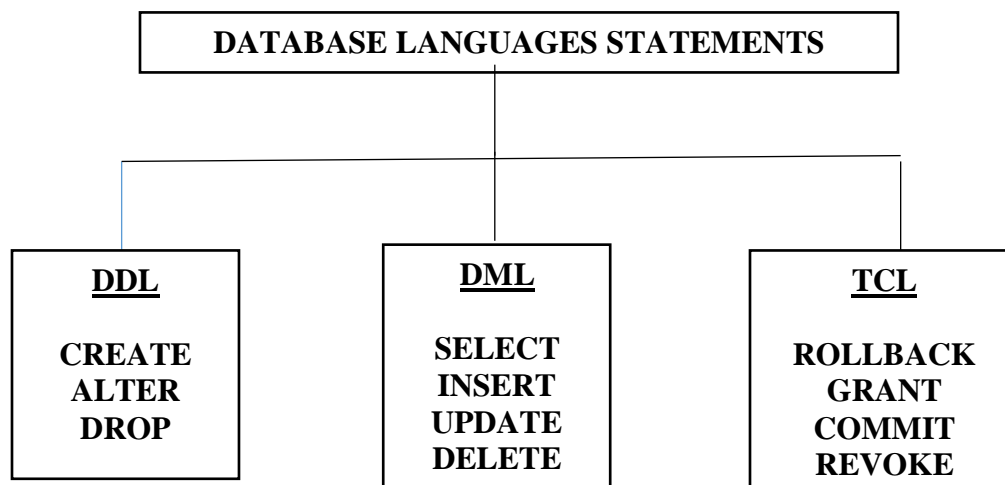


- SQL can retrieve data from a database
- SQL can insert new records in a database
- SQL can delete records from a database
- SQL can update records in a database
- SQL is easy to learn

6.2.1. NEED FOR SQL

SQL is an exceptional programming language that is utilized to interface with databases. It works by understanding and analyzing databases that include data fields in their tables. The primary objective where SQL was created was to give the possibility to common people get interested data from database. Structured Query Language (SQL) is used to retrieve data or otherwise interface with a relational database.

6.2.2. TYPES OF SQL STATEMENTS



SQL statements are categorized into four different type of statements, which are

1. DDL (DATA DEFINITION LANGUAGE)

A data definition or data description language (DDL) is syntax similar to a computer programming language for defining data structures, especially database schemas. DDL



statements create, modify, and remove database objects such as tables, indexes, and users. Common DDL statements are CREATE, ALTER, and DROP.

2. DML (DATA MANIPULATION LANGUAGE)

DML is abbreviation of Data Manipulation Language. It is used to retrieve, store, modify, delete, insert and update data in database. Examples: SELECT, UPDATE, INSERT statements. DDL is abbreviation of Data Definition Language. It is used to create and modify the structure of database objects in database.

3. TCL (TRANSACTION CONTROL LANGUAGE)

A data control language (DCL) is a syntax similar to a computer programming language used to control access to data stored in a database (Authorization). In particular, it is a component of Structured Query Language (SQL). Examples of DCL commands include: GRANT to allow specified users to perform specified tasks.

6.3. FEATURES OF SQL

SQL commands follow a number of basic rules:

- SQL keywords are not normally case sensitive, though this in this tutorial all commands (SELECT, UPDATE etc) are upper-cased.
- Variable and parameter names are displayed here as lower-case.
- New-line characters are ignored in SQL, so a command may be all on one line or broken up across a number of lines for the sake of clarity.
- Many DBMS systems expect to have SQL commands terminated with a semicolon character.

Some of the advantages of SQL are:

- With SQL, it is possible to query our database in several ways, using English-like statements.
- With SQL, a user can access data from a relational database management system.
- It allows the user to describe the data.



- It allows the user to define the data in the database and manipulate it when needed.
- It allows the user to create , drop database and table.
- It allows the user to create a view, stored procedure, function in a database.
- It allows the user to set permission on tables, procedures, and views.
- High speed. Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.
- No coding is needed.
- Well defined standards.
- Portability.

6.4. SELECT SQL OPERATIONS

The **select** statement is used to query the database and retrieve selected data that match the criteria that user specify.

Syntax:

```
select "column1" [,"column2",etc]  
from "tablename"  
[where "condition"];          [ ] = optional
```

The column names that follow the select keyword determine which columns will be returned in the results. The table name that follows the keyword **from** specifies the table that will be queried to retrieve the desired results. The **where** clause (optional) specifies which data values or rows will be returned or displayed, based on the criteria described after the keyword **where**.

Conditional selections used in the **where** clause:

- = Equal
- > Greater than
- < Less than
- >= Greater than or equal
- <= Less than or equal



◇ Not equal to

LIKE

The **LIKE** pattern matching operator can also be used in the conditional selection of the where clause. Like is a very powerful operator that allows you to select only rows that are "like" what you specify. The percent sign "%" can be used as a wild card to match any possible character that might appear before or after the characters specified.

Example:

1. select * from employee;

Empid	First	Last	Age	Address	City	State
1	Anu	Radha	45	Gandhi Nagar	Vellore	Tamil Nadu
2	Shanthi	Mani	30	Gandhi Nagar	Vellore	Tamil Nadu
3	Vidya	Lakshmi	30	Gandhi Nagar	Vellore	Tamil Nadu

2. select first, last, city from employee where first LIKE 'A%';

Empid	First	Last	Age	Address	City	State
1	Anu	Radha	45	Gandhi Nagar	Vellore	Tamil Nadu

This SQL statement will match any first names that start with 'A'. **Strings must be in single quotes.**

3. select first, last from employee where last LIKE '%a';

This statement will match any last names that end in 'a'.

Empid	First	Last	Age	Address	City	State
1	Anu	Radha	45	Gandhi Nagar	Vellore	Tamil Nadu

4. select * from employee where first = 'Vidya';



This will only select rows where the first name equals 'Vidya' exactly.

Empid	First	Last	Age	Address	City	State
3	Vidya	Lakshmi	30	Gandhi Nagar	Vellore	Tamil Nadu

6.5. GROUPING THE OUTPUT OF THE QUERY

6.5.1. SQL GROUP Functions

Group functions are built-in SQL functions that operate on groups of rows and return one value for the entire group. These functions are: **COUNT, MAX, MIN, AVG, SUM, DISTINCT**

a. SQL COUNT (): This function returns the number of rows in the table that satisfies the condition specified in the WHERE condition. If the WHERE condition is not specified, then the query returns the total number of rows in the table.

For Example: If you want the number of employees in a particular department, the query would be:

```
SELECT COUNT (*) FROM employee WHERE dept = 'Electronics';
```

The output would be '2' rows.

If you want the total number of employees in all the department, the query would take the form:

```
SELECT COUNT (*) FROM employee;
```

The output would be '5' rows.

b. SQL DISTINCT(): This function is used to select the distinct rows.

For Example: If you want to select all distinct department names from employee table, the query would be:

```
SELECT DISTINCT dept FROM employee;
```



To get the count of employees with unique name, the query would be:

```
SELECT COUNT (DISTINCT name) FROM employee;
```

c. **SQL MAX()**: This function is used to get the maximum value from a column.

To get the maximum salary drawn by an employee, the query would be:

```
SELECT MAX (salary) FROM employee;
```

d. **SQL MIN()**: This function is used to get the minimum value from a column.

To get the minimum salary drawn by an employee, he query would be:

```
SELECT MIN (salary) FROM employee;
```

e. **SQL AVG()**: This function is used to get the average value of a numeric column.

To get the average salary, the query would be

```
SELECT AVG (salary) FROM employee;
```

f. **SQL SUM()**: This function is used to get the sum of a numeric column

To get the total salary given out to the employees,

```
SELECT SUM (salary) FROM employee;
```

6.5.2. SQL GROUP BY CLAUSE

The SQL GROUP BY Clause is used along with the group functions to retrieve data grouped according to one or more columns.

For Example: If you want to know the total amount of salary spent on each department, the query would be:

```
SELECT dept, SUM (salary) FROM employee GROUP BY dept;
```



The output would be like:

dept	salary
Electrical	25000
Electronics	55000
Aeronautics	35000
InfoTech	30000

NOTE: The group by clause should contain all the columns in the select list except those used along with the group functions.

```
SELECT location, dept, SUM (salary) FROM employee GROUP BY location, dept;
```

The output would be like:

location	dept	salary
Bangalore	Electrical	25000
Bangalore	Electronics	55000
Mysore	Aeronautics	35000
Mangalore	InfoTech	30000

6.5.3. SQL HAVING Clause

Having clause is used to filter data based on the group functions. This is similar to WHERE condition but is used with group functions. Group functions cannot be used in WHERE Clause but can be used in HAVING clause.

SQL HAVING Clause Example

If you want to select the department that has total salary paid for its employees more than 25000, the sql query would be like;



```
SELECT dept, SUM (salary) FROM employee GROUP BY dept HAVING SUM (salary) > 25000;
```

The output would be like:

dept	salary
Electronics	55000
Aeronautics	35000
InfoTech	30000

When WHERE, GROUP BY and HAVING clauses are used together in a SELECT statement, the WHERE clause is processed first, then the rows that are returned after the WHERE clause is executed are grouped based on the GROUP BY clause.

Finally, any conditions on the group functions in the HAVING clause are applied to the grouped rows before the final output is displayed.

6.6. QUERYING FROM MULTIPLE TABLES

6.6.1. SQL SUBQUERY

Subquery or **Inner query** or **Nested query** is a query in a query. SQL subquery is usually added in the WHERE Clause of the SQL statement. Most of the time, a subquery is used when you know how to search for a value using a SELECT statement, but do not know the exact value in the database.

Subqueries are an alternate way of returning data from multiple tables.

Subqueries can be used with the following SQL statements along with the comparison operators like =, <, >, >=, <= etc.

- SELECT
- INSERT
- UPDATE
- DELETE



a. Subqueries with the SELECT Statement

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows

```
SELECT column_name [, column_name ] FROM table1 [, table2 ]
```

```
WHERE column_name OPERATOR
```

```
(SELECT column_name [, column_name ] FROM table1 [, table2 ] [WHERE])
```

Example

Consider the CUSTOMERS table having the following records

ID	NAME	AGE	ADDRES	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

Now, let us check the following subquery with a SELECT statement.

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

This would produce the following result.

ID	NAME	AGE	ADDRES	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00



b. Subqueries with the INSERT Statement

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

The basic syntax is as follows.

```
INSERT INTO table_name [ (column1 [, column2 ] ) ]  
SELECT [ *|column1 [, column2 ] FROM table1 [, table2 ] [WHERE VALUE OPERATOR ]
```

Example

Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS_BKP table, you can use the following syntax.

```
SQL> INSERT INTO CUSTOMERS_BKP SELECT * FROM CUSTOMERS WHERE ID IN  
(SELECT ID FROM CUSTOMERS) ;
```

c. Subqueries with the UPDATE Statement

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

The basic syntax is as follows.

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

Example

Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.



```
SQL> UPDATE CUSTOMERS SET SALARY = SALARY * 0.25 WHERE AGE IN (SELECT  
AGE FROM CUSTOMERS_BKP WHERE AGE >= 27);
```

This would impact two rows and finally CUSTOMERS table would have the following records.

ID	NAME	AGE	ADDRES	SALARY
1	Ramesh	35	Ahmedabad	125.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	2125.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

d. Subqueries with the DELETE Statement

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic syntax is as follows.

```
DELETE FROM TABLE_NAME [WHERE OPERATOR [ VALUE ]  
(SELECT COLUMN_NAME FROM TABLE_NAME) [WHERE]]
```

Example

Assuming, we have a CUSTOMERS_BKP table available which is a backup of the CUSTOMERS table. The following example deletes the records from the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
SQL> DELETE FROM CUSTOMERS WHERE AGE IN (SELECT AGE FROM  
CUSTOMERS_BKP WHERE AGE >= 27 );
```



This would impact two rows and finally the CUSTOMERS table would have the following records.

ID	NAME	AGE	ADDRES	SALARY
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

6.6.2. CORRELATED SUBQUERY

A query is called correlated subquery when both the inner query and the outer query are interdependent. For every row processed by the inner query, the outer query is processed as well. The inner query depends on the outer query before it can be processed.

```
SELECT p.product_name FROM product p WHERE p.product_id = (SELECT  
o.product_id FROM order_items o WHERE o.product_id = p.product_id);
```

NESTED SUBQUERY

1) You can nest as many queries you want but it is recommended not to nest more than 16 subqueries in oracle

NON-CORELATED SUBQUERY

2) If a subquery is not dependent on the outer query it is called a non-correlated subquery

SUBQUERY ERRORS

3) Minimize subquery errors: Use drag and drop, copy and paste to avoid running subqueries with spelling and database typos. Watch your multiple field SELECT comma use, extra or to few getting SQL error message "Incorrect syntax".



SQL SUBQUERY COMMENTS

Adding SQL Subquery comments are good habit (/* your command comment */) which can save you time, clarify your previous work. results in less SQL headaches

6.7. RETRIEVAL USING SET OPERATORS

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

There are 4 different types of SET operations, along with example:

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

6.7.1. UNION Operation

UNION is used to combine the results of two or more **SELECT** statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.

Example of UNION

The **First** table,

ID	Name
1	abhi
2	adam

The **Second** table,

ID	Name
2	adam
3	Chester

Union SQL query will be,



```
SELECT * FROM First  
UNION  
SELECT * FROM Second;
```

The result set table will look like,

ID	NAME
1	abhi
2	adam
3	Chester

6.7.2. UNION ALL

This operation is similar to Union. But it also shows the duplicate rows.

Example of Union All

The **First** table,

ID	NAME
1	abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester

Union All query will be like,

```
SELECT * FROM First  
UNION ALL  
SELECT * FROM Second;
```



The resultset table will look like,

ID	NAME
1	abhi
2	adam
2	adam
3	Chester

6.7.3. INTERSECT

Intersect operation is used to combine two `SELECT` statements, but it only returns the records which are common from both `SELECT` statements. In case of **Intersect** the number of columns and datatype must be same.

NOTE: MySQL does not support INTERSECT operator.

Example of Intersect

The **First** table,

ID	NAME
1	abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester



Intersect query will be,
SELECT * FROM First
INTERSECT
SELECT * FROM Second;

The resultset table will look like

ID	NAME
2	adam

6.7.4. MINUS

The Minus operation combines results of two `SELECT` statements and return only those in the final result, which belongs to the first set of the result.

Example of Minus

The **First** table,

ID	NAME
1	abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester



Minus query will be,

```
SELECT * FROM First
```

```
MINUS
```

```
SELECT * FROM Second;
```

The resultset table will look like,

ID	NAME
1	abhi

REVIEW QUESTIONS

1. Discuss in detail the operators SELECT, PROJECT and UNION with suitable examples.
2. Explain about the following clauses with example queries.
 - (i) Group by
 - (ii) Order by
 - (iii) Aggregation functions.
3. Give syntaxes to Create and Alter a table.
4. List aggregate functions supported by SQL.
5. Explain the role of views. Why role got importance? What are the problems in view updating?
6. Give syntax for DML commands? Show their operations with an example?
7. List and Explain SET operations of SQL.
8. Where do we need nesting of queries? Give an example.



9. Differentiate between updatable views and non-updatable views?
10. Write string operations supported by SQL.
11. Consider the following relation schema:
Sailors (sid: integer, sname: string, rating: integer, age: real)
Boat (bid: integer, bname: string, color: string)
Reserves (sid: integer, bid: integer, day: date)
Write the following queries in SQL.
 - a. Find the average age of the sailor who are eligible for voting for each rating level that has at least two sailors.
 - b. Find the name of sailors who have reserved both red and a green boat.
 - c. Find the sailor_id of sailors who have reserved a red boat
12. Discuss the following clauses with examples
 - (i) HAVING
 - (ii) GROUP BY
 - (iii) Relational set operations.
13. Identify some informal queries and update operations that you would expect to apply to the database shown in Figure 1.
14. Specify all the relationships among the records of the database shown in Figure 1.
15. Give some additional views that may be needed by other user groups for the database shown in Figure 1.
16. Cite some examples of integrity constraints that you think can apply to the database shown in Figure 1.
17. Consider Figure 1.
 - A. If the name of the 'CS' (Computer Science) Department changes to 'CSSE' (Computer Science and Software Engineering) Department and the corresponding prefix for the course number also changes, identify the columns in the database that would need to be updated.
 - B. Can you restructure the columns in the COURSE, SECTION, and PREREQUISITE tables so that only one column will need to be updated?



STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

Figure 1:

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

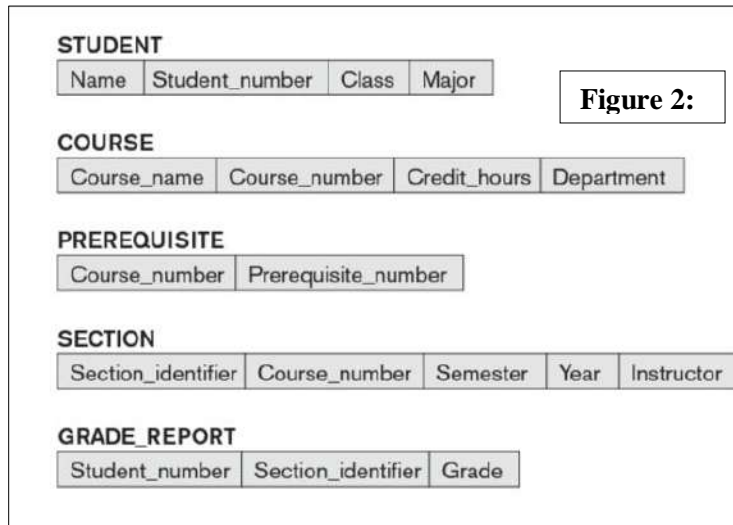
Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310



18. Think of different users for the database shown in Figure 1. What types of applications would each user need? To which user category would each belong, and what type of interface would each need?
19. Choose a database application with which you are familiar. Design a schema and show a sample database for that application, using the notation of Figures 1. and 2. What types of additional information and constraints would you like to represent in the schema? Think of several users of your database, and design a view for each.
20. Consider Figure 2.

In addition to constraints relating the values of columns in one table to columns in another table, there are also constraints that impose restrictions on values in a column or a combination of columns within a table. One such constraint dictates that a column or a group of columns must be unique across all rows in the table.

For example, in the STUDENT table, the Student_number column must be unique (to prevent two different students from having the same Student_number). Identify the column or the group of columns in the other tables that must be unique across all rows in the table.



21. How do the relations (tables) in SQL differ from the relations defined formally in Relational Database? Discuss the other differences in terminology. Why does SQL allow duplicate tuples in a table or in a query result?
22. List the data types that are allowed for SQL attributes.
23. Describe the four clauses in the syntax of a simple SQL retrieval query. Show what type of constructs can be specified in each of the clauses. Which are required and which are optional?
24. Consider the database shown in Figure 1. Whose schema is shown in Figure 2? What are the referential integrity constraints that should hold on the schema? Write appropriate SQL DDL statements to define the database.
25. Write SQL update statements to do the following on the database schema shown in Figure 1.
 - a. Insert a new student, <'Johnson', 25, 1, 'Math'>, in the database.
 - b. Change the class of student 'Smith' to 2.
 - c. Insert a new course, <'Knowledge Engineering', 'CS4390', 3, 'CS'>.
 - d. Delete the record for the student whose name is 'Smith' and whose student number is 17.
26. Design a relational database schema for a database application of your choice.
 - a. Declare your relations, using the SQL DDL.
 - b. Specify a number of queries in SQL that are needed by your database application.
 - c. Based on your expected use of the database, choose some attributes that should have indexes specified on them.
 - d. Implement your database, if you have a DBMS that supports SQL.
27. Describe the six clauses in the syntax of an SQL retrieval query. Show what type of constructs can be specified in each of the six clauses. Which of the six clauses are required and which are optional?
28. Describe conceptually how an SQL retrieval query will be executed by specifying the conceptual order of executing each of the six clauses.



29. Discuss how NULLs are treated in comparison operators in SQL. How are NULLs treated when aggregate functions are applied in an SQL query? How are NULLs treated if they exist in grouping attributes?
30. Discuss how each of the following constructs is used in SQL, and discuss the various options for each construct. Specify what each construct is useful for.
- Nested queries.
 - Joined tables and outer joins.
 - Aggregate functions and grouping.
 - Triggers.
 - Assertions and how they differ from triggers.
 - Views and their updatability.
 - Schema change commands.

Figure 5:

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

DEPT_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

WORKS_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

31. Specify the following queries in SQL on the database schema in Figure 1.



- a. Retrieve the names and major departments of all straight-A students (students who have a grade of A in all their courses).
 - b. Retrieve the names and major departments of all students who do not have a grade of A in any of their courses.
32. In SQL, specify the following queries on the database in Figure 5 using the concept of nested queries.
- a. Retrieve the names of all employees who work in the department that has the employee with the highest salary among all employees.
 - b. Retrieve the names of all employees whose supervisor's supervisor has '888665555' for SSN.
 - c. Retrieve the names of employees who make at least \$10,000 more than the employee who is paid the least in the company.
33. Specify the following views in SQL on the COMPANY database schema shown in Figure 3.
- a. A view that has the department name, manager name, and manager salary for every department.
 - b. A view that has the employee name, supervisor name, and employee salary for each employee who works in the 'Research' department.
 - c. A view that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project.
 - d. A view that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project *with more than one employee working on it*.
34. Consider the following view, DEPT_SUMMARY, defined on the COMPANY database in Figure 3.



CREATE VIEW DEPT_SUMMARY (D, C, Total_s, Average_s) **AS**
SELECT Dno, **COUNT** (*), **SUM** (Salary), **AVG** (Salary) **FROM**
EMPLOYEE GROUP BY Dno;

State which of the following queries and updates would be allowed on the view. If a query or update would be allowed, show what the corresponding query or update on the base relations would look like, and give its result when applied to the database in Figure 3.

- a. **SELECT * FROM** DEPT_SUMMARY;
- b. **SELECT** D, C **FROM** DEPT_SUMMARY **WHERE** TOTAL_S > 100000;
- c. **SELECT** D, AVERAGE_S **FROM** DEPT_SUMMARY **WHERE** C > (
SELECT C **FROM** DEPT_SUMMARY **WHERE** D=4);
- d. **UPDATE** DEPT_SUMMARY **SET** D=3 **WHERE** D=4;
- e. **DELETE FROM** DEPT_SUMMARY **WHERE** C > 4;

35. List the operations of relational algebra and the purpose of each.
36. What is union compatibility? Why do the UNION, INTERSECTION, and DIFFERENCE operations require that the relations on which they are applied be union compatible?
37. Discuss some types of queries for which renaming of attributes is necessary in order to specify the query unambiguously.
38. How are the OUTER JOIN operations different from the INNER JOIN operations? How is the OUTER UNION operation different from UNION?
39. List the various cases where use of a NULL value would be appropriate.
40. Define the following terms: *entity*, *attribute*, *attribute value*, *relationship instance*, *composite attribute*, *multivalued attribute*, *derived attribute*, *complex attribute*, *key attribute*, and *value set (domain)*.



41. Under what conditions can an attribute of a binary relationship type be migrated to become an attribute of one of the participating entity types?
42. Consider an entity type SECTION in a UNIVERSITY database, which describes the section offerings of courses.

The attributes of SECTION are Section_number, Semester, Year, Course_number, Instructor, Room_no (where section is taught), Building (where section is taught), Weekdays (domain is the possible combinations of weekdays in which a section can be offered {'MWF', 'MW', 'TT', and so on}), and Hours (domain is all possible time periods during which sections are offered {'9–9:50 A.M.', '10–10:50 A.M.', ..., '3:30–4:50 P.M.', '5:30–6:20 P.M.', and so on}).

Assume that Section_number is unique for each course within a particular semester/year combination (that is, if a course is offered multiple times during a particular semester, its section offerings are numbered 1, 2, 3, and so on). There are several composite keys for section, and some attributes are components of more than one key. Identify three composite keys, and show how they can be represented in an ER schema diagram.

43. Consider the following GRADEBOOK relational schema describing the data for a grade book of a particular instructor. (*Note:* The attributes A, B, C, and D of COURSES store grade cutoffs.)

CATALOG(Cno, Ctitle)
STUDENTS(Sid, Fname, Lname, Minit)
COURSES(Term, Sec_no, Cno, A, B, C, D)
ENROLLS(Sid, Term, Sec_no)

Specify and execute the following queries using the RA interpreter on the GRADEBOOK database schema.



- Retrieve the names of students enrolled in the Automata class during the fall 2009 term.
- Retrieve the Sid values of students who have enrolled in CSc226 and CSc227.
- Retrieve the Sid values of students who have enrolled in CSc226 or CSc227.
- Retrieve the names of students who have not enrolled in any class.
- Retrieve the names of students who have enrolled in all courses in the CATALOG table.

OBJECTIVE TYPE QUESTIONS

1. Which SQL Query is use to remove a table and all its data from the database?

- | | |
|-----------------|------------------|
| a) Create Table | c) Drop Table |
| b) Alter Table | d) None of these |

2. In precedence of set operators the expression is evaluated from:

- | | |
|------------------|-------------------|
| a) Left to Left | c) Right to Right |
| b) Left to Right | d) Right to Left |

3. A logical description of some portion of database that is required by a user to perform task is called as

- | | |
|----------------|-----------------|
| a) System View | c) Logical View |
| b) User View | d) Data View |

4. A command to remove a relation from an SQL database

- | | |
|------------------------------|-----------------------------|
| a) Delete table <table name> | c) Erase table <table name> |
| b) Drop table <table name> | d) Alter table <table name> |

5. Which of the following is not an Aggregate function?

- | | |
|--------|-----------|
| a) Min | c) Select |
| b) Max | d) Avg |

6. Which of the following is not Modification of the Database

- | | |
|--------------|-------------|
| a) Deletion | c) Sorting |
| b) Insertion | d) Updating |



7. A type of query that is placed within a WHERE or HAVING clause of another query is called

- a) Super query
- b) Sub query
- c) Master query
- d) Multi-query

8. A transaction completes its execution is said to be

- a) Saved
- b) Loaded
- c) Rolled
- d) Committed

9 Which of the following is correct regarding Aggregate functions?

- a) it takes a list of values and return a single values as result
- b) it takes a list of values and return a list of values as result
- c) it takes a single value and returns a list of values as result
- d) it takes a single value and returns a single value as result

10. Which of the following operation is used if we are interested in only certain columns of a table?

- a. PROJECTION
- b. SELECTION
- c. UNION
- d. JOIN

11. Which of the following is a comparison operator in SQL?

- a) =
- b) LIKE
- c) BETWEEN
- d) All of the above

12. To delete a particular column in a relation the command used is:

- a) UPDATE
- b) DROP
- c) ALTER
- d) DELETE

13. The _____ operator is used to compare a value to a list of literals values that have been specified.

- a) BETWEEN
- b) ANY
- c) IN
- d) ALL

14. Which of the following is a valid SQL type?

- a) CHARACTER
- b) NUMERIC
- c) FLOAT
- d) All of the above



15. Count function in SQL returns the number of

- a. values.
- b. distinct values.
- c. groups.
- d. columns

16. _____ is a virtual table that draws its data from the result of an SQL SELECT statement.

- a) View
- b) Synonym
- c) Sequence
- d) Transaction

KEYS

1-c, 2-b, 3-b, 4-b, 5-c, 6-c, 7-b, 8-d, 9-a, 10-a, 11-d, 12-c, 13-a, 14-d,
15-a, 16-a



CHAPTER 7: T-SQL – TRIGGERS AND DYNAMIC EXECUTION

7.1. INTRODUCTION

Oracle uses an extension of SQL called PL/SQL whereas MS SQL Server/Sybase uses Trans-SQL. Transact-SQL enhances the power of SQL and minimizes the occasions on which users must resort to a programming language to accomplish a desired task. Extended-SQL's capabilities go beyond the many commercial versions of SQL.

T-SQL is organized by each block of statement. A block of statement can embrace another block of statement in it. A block of statement starts by BEGIN and finishes by END. There are many statements in the block, and statements is separated from each other by a semicolon (;).

The structure of the block:

?

- 1 BEGIN
- 2 -- Declare variables
- 3 -- T-SQL Statements
- 4 END;

7.2. T-SQL (TRANSACT-SQL)

T-SQL (Transact-SQL) is a set of programming extensions from Sybase and Microsoft that add several features to the Structured Query Language (SQL), including transaction control, exception and error handling, row processing and declared variables.

All applications that communicate with SQL Server do so by sending T-SQL statements to the server. T-SQL queries include the SELECT statement, selecting columns, labeling output columns, restricting rows and modifying a search condition.



T-SQL identifiers, meanwhile, are used in all databases, servers, and database objects in SQL Server. These include the following tables, constraints, stored procedures, views, columns and data types. T-SQL identifiers must each have a unique name, are assigned when an object is created and are used to identify an object.

T-SQL STATEMENT EXAMPLES

The most popular T-SQL statement is the stored procedure, which is a compiled and stored T-SQL code. Similar to views, stored procedures generate an execution plan when called the first time. The difference is stored procedures can select data and execute any T-SQL code within any parameters.

User-defined functions are another example of T-SQL statements. User-defined functions take input parameters, perform an action and return the results to the call.

RESTORE A DATABASE WITH T-SQL.

Another example is a trigger, which is a stored T-SQL script that runs when a statement other than SELECT is issued against a table or view. The two common triggers are AFTER triggers and INSTEAD OF triggers.

Programming T-SQL statements enables IT pros to build applications contained within SQL Server. These applications -- or objects -- can insert, update, delete or read data stored in a database.

7.2.1. T-SQL FUNCTIONS

In addition to SQL Server's built-in functions, users can define functions using T-SQL.

- Types of T-SQL functions include:
- Aggregate functions, which operate on a collection of values, but return one summary value.
- Ranking functions, which return a ranking value for every row within a partition.



- Rowset functions, which return an object that can be used as a table reference in SQL statements.
- Scalar functions, which operate on a single value and return a single value.

SQL Server also supports analytical functions in T-SQL to depict complex analytical tasks. These analytical functions enable IT pros to perform common analysis, such as ranking, percentiles, moving averages and cumulative sums to be expressed in a single SQL statement.

DIFFERENCE BETWEEN T-SQL AND SQL

There are three distinct differences between the two.

- While T-SQL is an extension to SQL, SQL is a programming language.
- T-SQL contains procedural programming and local variable, while SQL does not.
- T-SQL is proprietary, while SQL is an open format.

JOINS IN T-SQL

Joins in T-SQL are clauses used to combine rows from two or more tables, based on a related column between them. Joins specify how SQL should use data from one table to select the rows in another table. Several operators -- such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT -- can be used to join tables.

Different types of joins are available in T-SQL. They include, for example, *inner joins* and *outer joins*. An inner join, which returns rows when there is a match in both tables, can be specified in either the FROM or WHERE clauses. Outer joins, which can be specified in the FROM clause only, finds and returns matching data and some dissimilar data from tables.



7.2.2. TRANSACT-SQL

Transact-SQL is a database procedural programming language. Microsoft's monopoly, used in **SQL Server**. Procedural languages are designed to extend SQL's abilities while being able to integrate well with **SQL**. Several features such as local variables and string/data processing are added. These features make the language Turing-complete. They are also used to write stored procedures: pieces of code residing on the server to manage complex business rules that are hard or impossible to manage with pure set-based operations.

A Turing Complete system means a system in which a program can be written that will find an answer (although with no guarantees regarding runtime or memory).

REVIEW QUESTIONS

1. Mention what is T-SQL?
2. Mention what is the difference between SQL and T-SQL?
3. Please name at least five commands which can manipulate text in the T-SQL code. For example, replace a text string, obtain a portion of the text, etc.
4. Is it possible to import data directly from T-SQL commands without using SQL Server Integration Services? If so, what are the commands?
5. Mention new error handling commands which are introduced with the SQL Server 2005 and beyond? What commands did they replace? How are they command used?
6. Mention how T-SQL statements can be written and submitted to the Database engine?
7. Mention what is "GO" in T-SQL?
8. Difference between Delete & Truncate Statement? Which Statement Can Be Rolledback?
9. Why you should not use a Cursor? What Are Its Alternatives?
10. What Are The Multiple Ways To Execute A Dynamic Query?
11. What should be the Ideal Combination with in & Union (all) in terms of Performance?



UNIT – V

CHAPTER 8: PROCEDURAL LANGUAGE

8.1. INTRODUCTION

PL/SQL is an extension of Structured Query Language (SQL) that is used in Oracle. Unlike SQL, PL/SQL allows the programmer to write code in a procedural format. Full form of PL/SQL is "Procedural Language extensions to SQL". It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

PL/SQL means instructing the compiler 'what to do' through SQL and 'how to do' through its procedural way. Similar to other database languages, it gives more control to the programmers by the use of loops, conditions and object-oriented concepts.



Figure 8.1: Features of PL/SQL

FOLLOWING ARE CERTAIN NOTABLE FACTS ABOUT PL/SQL

- PL/SQL is a completely portable, high-performance transaction-processing language.



- PL/SQL provides a built-in, interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the command-line **SQL*Plus interface**.
- Direct call can also be made from external programming language calls to database.
- PL/SQL's general syntax is based on that of ADA and Pascal programming language.
- Apart from Oracle, PL/SQL is available in **Times Ten in-memory database** and **IBM DB2**.

ADVANTAGES OF PL/SQL

PL/SQL is a completely portable, high-performance transaction processing language that offers the following advantages:

- Tight Integration with SQL
- Better Performance
- Higher Productivity Full Portability
- Tight Security
- Access to Pre-defined Packages
- Support for Object-Oriented Programming
- Support for Developing Web Applications and Pages

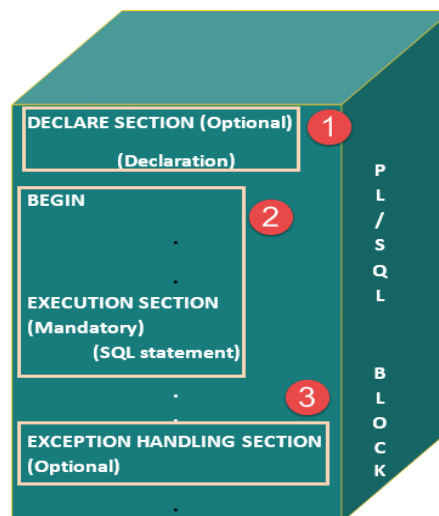
8.2. PL/ SQL BLOCK STRUCTURE

The basic unit of a PL/SQL source program is the block, which groups related declarations and statements. A PL/SQL block is defined by the keywords **DECLARE, BEGIN, EXCEPTION, and END**. These keywords partition the block into a declarative part, an executable part, and an exception-handling part. Only the executable part is required.



Declarations are local to the block and cease to exist when the block completes execution, helping to avoid cluttered namespaces for variables and subprograms. Blocks can be nested: Because a block is an executable statement, it can appear in another block wherever an executable statement is allowed.

In PL/SQL, the code is not executed in single line format, but it is always executed by grouping the code into a single element called Blocks. In this tutorial, you are going to learn about these blocks.



The above picture illustrates the different PL/SQL block and their section order.

Blocks contain both PL/SQL as well as SQL instruction. All these instruction will be executed as a whole rather than executing a single instruction at a time. PL/SQL blocks have a pre-defined structure in which the code is to be grouped.

Below are different sections of PL/SQL blocks.

1. Declaration section
2. Execution section
3. Exception-Handling section



1. DECLARATION SECTION

This is the first section of the PL/SQL blocks. This section is an optional part. This is the section in which the declaration of variables, cursors, exceptions, subprograms, pragma instructions and collections that are needed in the block will be declared.

Below are few more characteristics of this part.

- This particular section is optional and can be skipped if no declarations are needed.
- This should be the first section in a PL/SQL block, if present.
- This section starts with the keyword 'DECLARE' for triggers and anonymous block. For other subprograms, this keyword will not be present. Instead, the part after the subprogram name definition marks the declaration section.
- This section should always be followed by execution section.

2. EXECUTION SECTION

Execution part is the main and mandatory part which actually executes the code that is written inside it. Since the PL/SQL expects the executable statements from this block this cannot be an empty block, i.e., it should have at least one valid executable code line in it.

Below are few more characteristics of this part.

- This can contain both PL/SQL code and SQL code.
- This can contain one or many blocks inside it as a nested block.
- This section starts with the keyword 'BEGIN'.
- This section should be followed either by 'END' or Exception-Handling section (if present)

3. EXCEPTION-HANDLING SECTION

The exception is unavoidable in the program which occurs at run-time and to handle this Oracle has provided an Exception-handling section in blocks. This section can also contain PL/SQL statements. This is an optional section of the PL/SQL blocks.

- This is the section where the exception raised in the execution block is handled.



- This section is the last part of the PL/SQL block.
- Control from this section can never return to the execution block.
- This section starts with the keyword 'EXCEPTION'.
- This section should always be followed by the keyword 'END'.
- The Keyword 'END' marks the end of PL/SQL block.

8.2.1. PL/SQL BLOCK SYNTAX

Below is the syntax of the PL/SQL block structure.

```
DECLARE --optional
    <declarations>
BEGIN --mandatory
    <executable statements. At least one executable statement is mandatory>
EXCEPTION --optional
    <exception handles>
END; --mandatory
/
```

Note: A block should always be followed by '/' which sends the information to the compiler about the end of the block.

8.2.2. TYPES OF PL/SQL BLOCK

PL/SQL blocks are of mainly two types.

1. Anonymous blocks
2. Named Blocks



1. ANONYMOUS BLOCKS

Anonymous blocks are PL/SQL blocks which do not have any names assigned to them. They need to be created and used in the same session because they will not be stored in the server as database objects.

Since they need not store in the database, they need no compilation steps. They are written and executed directly, and compilation and execution happen in a single process.

Below are few more characteristics of Anonymous blocks.

- These blocks don't have any reference name specified for them.
- These blocks start with the keyword 'DECLARE' or 'BEGIN'.
- Since these blocks do not have any reference name, these cannot be stored for later purpose. They shall be created and executed in the same session.
- They can call the other named blocks, but call to anonymous block is not possible as it is not having any reference.
- It can have nested block in it which can be named or anonymous. It can also be nested in any blocks.
- These blocks can have all three sections of the block, in which execution section is mandatory, the other two sections are optional.

EXAMPLE:

```
1 DECLARE
2 num NUMBER(2);
3 sq NUMBER(3);
4 BEGIN
5 num:= &Number1;
```



```
6 sq := num*num;
7 DBMS_OUTPUT.PUT_LINE('Square:' ||sq);
8 END;
```

2. NAMED BLOCKS

Named blocks have a specific and unique name for them. They are stored as the database objects in the server. Since they are available as database objects, they can be referred to or used as long as it is present on the server. The compilation process for named blocks happens separately while creating them as a database objects.

Below are few more characteristics of Named blocks.

- These blocks can be called from other blocks.
- The block structure is same as an anonymous block, except it will never start with the keyword 'DECLARE'. Instead, it will start with the keyword 'CREATE' which instruct the compiler to create it as a database object.
- These blocks can be nested within other blocks. It can also contain nested blocks

EXAMPLE:

```
1 FUNCTION sqr (num IN NUMBER)
2 RETURN NUMBER is sq NUMBER(2);
3 BEGIN
4 sq:= num*num;
5 RETURN sq;
6 END;
```




8.2.3. FUNDAMENTALS OF PL/SQL

PL/SQL IDENTIFIERS

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

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

PL/SQL DELIMITERS

A delimiter is a symbol with a special meaning. Following is the list of delimiters in PL/SQL:

Delimiter	Description
+, -, *, /	Addition, subtraction/negation, multiplication, division
%	Attribute indicator
'	Character string delimiter
.	Component selector
(,)	Expression or list delimiter
:	Host variable indicator
,	Item separator
"	Quoted identifier delimiter
=	Relational operator
@	Remote access indicator



;	Statement terminator
:=	Assignment operator
=>	Association operator
	Concatenation operator
**	Exponentiation operator
<<, >>	Label delimiter (begin and end)
/*, */	Multi-line comment delimiter (begin and end)
--	Single-line comment indicator
..	Range operator
<, >, <=, >=	Relational operators
<>, !=, ~=, ^=	Different versions of NOT EQUAL

PL/SQL COMMENTS

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

The PL/SQL supports single-line and multi-line comments. All characters available inside any comment are ignored by the PL/SQL compiler. The PL/SQL single-line comments start with the delimiter -- (double hyphen) and multi-line comments are enclosed by /* and */.



DECLARE

-- variable declaration

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

BEGIN

```
/*
```

```
* PL/SQL executable statement(s)
```

```
*/
```

```
dbms_output.put_line(message);
```

```
END;
```

```
/
```

OUTPUT:

```
Hello World
```

```
PL/SQL procedure successfully completed.
```

8.3. PL/SQL - DATA TYPES

The PL/SQL variables, constants and parameters must have a valid data type, which specifies a storage format, constraints, and a valid range of values. There are six built-in PL/SQL data types:

1. Scalar data types - Scalar data types haven't internal components.
2. Composite data types - Composite data types have internal components to manipulate data easily.
3. Reference data types - This data types work like a pointer to hold some value.
4. LOB data types - Stores large objects such as images, graphics, video.
5. Unknown Column types - Identify columns when not know the type of column.



6. User Define data types - Define your own data type that inherited from predefined base data type.

8.3.1. SCALAR DATA TYPES

Scalar data types haven't internal components. It is like a linear data type.

Scales data type divides into four different types

1. Numeric
2. Character
3. Boolean
4. Date/Time

a. NUMERIC DATA TYPES

Following are numeric data types in PL/SQL:

Datatype	Description, Storage(Maximum)
NUMBER(p,s)	NUMBER data type used to store numeric data.
BINARY_INTEGER	BINARY_INTEGER data type store signed integer's value.
PLS_INTEGER	PLS_INTEGER data type used to store signed integers data.

b. CHARACTER DATA TYPES

Character Data types used to store an alphabetic/alphanumeric character. Following are some character data types in PL/SQL

Datatype	Description	Storage (Maximum)
CHAR	CHAR data type used to store character data within a predefined length.	32767 bytes



CHARACTER	CHARACTER data type same as CHAR data type. It is another name of CHAR data type.	32767 bytes
VARCHAR2	VARCHAR2 data type used to store variable strings data within a predefined length.	32767 bytes
LONG	LONG data type used to store variable string data within a predefined length, This data type used for backward compatibility. Please use LONG data to the CLOB type.	32760 bytes

c. BOOLEAN DATA TYPES

Boolean Data Types stores logical values either TRUE or FALSE

Datatype	Description
Boolean	Boolean data type stores logical values. Boolean data types doesn't take any parameters. Boolean data type store, either TRUE or FALSE. Also, store NULL, Oracle treats NULL as an unassigned boolean variable.

d. DATE/TIME DATATYPES

A variable that has date/time data type hold value call datetimes. Oracle SQL automatically converts character value into default date format ('DD-MON-YY') TO_DATE values.

Following are Date/Time data types in Oracle SQL.

Datatype	Description
DATE	DATE data type to store valid date-time format with a fixed length.



TIMESTAMP	TIMESTAMP data type to store valid date (year, month, day) with time (hour, minute, second).
-----------	--

In PL/SQL datetime data type or interval data type fields values show the valid values for each field.

Field Name	Valid Value	Valid Interval Value
YEAR	-4712 to 9999	IntegerValue exclude 0
MONTH	01 to 12	0 to 11
DAY	01 to 31	Integer Value exclude 0
HOUR	00 to 23	0 to 23
MINUTE	00 to 59	0 to 59
SECOND	00 to 59.9(n) here n is precision of time fractional seconds	0 to 59.9(n)

8.3.2. PL/SQL COMPOSITE DATA TYPES

A composite data type stores values that have internal components and internal components can be either scalar or composite. Internal components can be of same data type and different data type. PL/SQL allows us to define two kinds of composite data types:

1. Collection - The internal components must have the same data type and we can access each element of a collection variable by its unique index

syntax: variable_name(index).



- Record - The internal components can have different data types and we can access each field of a record variable by its name

syntax: variable_name.field_name.

Collections in PL/SQL:

Oracle provides three types of collections.

- Index-by Table(associate array),
- Nested Tables, and
- VARRAY.

All these collections are like a single dimension array.

Syntax of collection declaration is as follows:-

TYPE type IS -- type is collection variable name, a valid identifier

```
{ assoc_array_type_def  
| varray_type_def  
| nested_table_type_def  
};
```

Collection can be created in following ways

- Defines a collection type and then declare a variable of that type.
- Use %TYPE to declare a collection variable of the same type as a previously declared collection variable.

ASSOCIATE ARRAY (INDEXED TABLES):

Associative array is a set of key-value pairs and each key should be unique index. The data type of index can be either a string type or PLS_INTEGER. Indexes are stored in sort order, not creation order.

Syntax of associative array type creation :

```
TYPE type IS {  
--assoc_array_type_def
```



TABLE OF datatype [NOT NULL]

```
INDEX BY { PLS_INTEGER | BINARY_INTEGER | VARCHAR2 ( v_size ) |  
data_type }  
};
```

On combining both collection declaration and associate table type declaration, we create associative array and store key value pairs in following program and we can perform various operation on it (Collections method).It's first way of creating collection(another way uses %TYPE).In below sample program, we create TYPE of associative array named as address and then create a variable employee_address of TYPE address. Refer in-line comments for more details:

DECLARE

```
--Associative array type indexed by BINARY_NUMBER  
TYPE address IS TABLE OF VARCHAR2(200) INDEX BY BINARY_INTEGER ;  
--Associative array variable of type address  
employees_address address;
```

BEGIN

```
employees_address('01') := 'Hyderabad, INDIA';  
employees_address('02') := 'Banglore, INDIA';  
employees_address('03') := 'NY, USA';  
-- FIRST and NEXT gives firs and next element of collecton  
DBMS_OUTPUT.PUT_LINE('FIRST and LAST ELEMENT key of collection are ' ||  
employees_address.FIRST || ' and ' || employees_address.LAST);  
--COUNT()method gives total no of elements in collection  
DBMS_OUTPUT.PUT_LINE('Total no of elements in collection '  
|| employees_address.COUNT);  
--EXISTS check for existence of key  
IF employees_address.EXISTS(02) THEN  
employees_address.DELETE(02);
```




END IF;

```
DBMS_OUTPUT.PUT_LINE('Total no of elements in collection after delete '  
|| employees_address.COUNT);
```

END;

OUTPUT:

FIRST and LAST ELEMENT key of collection are 1 and 3

Total no of elements in collection 3

Total no of elements in collection after delete 2

VARRAY:

It is variable-size array and element counts in it can vary from 0 to declared maximum size.

Characteristics of VARRAY:

- Elements of VARRAY can be accessed by variable_name (index). VARRAY index starts from 1 (lowest_index = 1) and it can go up to maximum size of VARRAY.
- As contrast to associative array, **it can be persisted in database table** and order of elements (indexes and element order) remain stable.
- VARRAY has constructor support as contrast to Associative array that does not support collection constructor. A **collection constructor** is a system-defined function with the **same name as a collection type**, which returns a collection of that type. Syntax of a constructor invocation is:

```
collection_type ([values,...]), values are optional. If no value is passed  
constructor returns empty collection.
```
- VARRAY is stored as a single object in a column in database table.(if size of object is more than 4KB then it is stored separately but in same namespace).
Following diagram depicts how VARRAY is stored in database table: Highlighted column refers to VARRAY type and stored in database column as other scalar type.



Employees table in database with a column of VARRAY type: (Local/Permanent/and other address)

EMP_ID	Name	EMPLOYEE_ADDRESS			
1	MIKE	NY, USA	AUSTIN, USA		
2	RITZ	HYD, INDIA	BANG, INDIA		

VARRAY creation and its initialization

Syntax of VARRAY creation is as follows - varray_type_def with collection

-- size_limit: upper limit of VARRAY(maximum that many elements can be stored)

TYPE type IS { VARRAY | [VARYING] ARRAY } (size_limit)

OF datatype [NOT NULL]

Consider following sample program which creates a VARRAY to store address information of employees and initialize it with constructor. Here ADDRESS is VARRAY type with upper limit of container 3 and using constructor collection of type ADDRESS created is returned to emp_address.

DECLARE

-- VARRAY type declaration of type VARCHAR, upperlimit 3

TYPE ADDRESS IS VARRAY(3) OF VARCHAR2(45);

-- varray variable initialized with constructor of type ADDRESS

emp_address ADDRESS := ADDRESS('HYD,IND', 'NY,USA','BANG,IND');

BEGIN

DBMS_OUTPUT.PUT_LINE('VARRAY elements count is '

|| emp_address.COUNT);

DBMS_OUTPUT.PUT_LINE('Address display - Iteration over VARRAY');

--emp_address.FIRST= 1 and emp_address.LAST = 3



```
FOR i IN emp_address.FIRST..emp_address.LAST LOOP
```

```
DBMS_OUTPUT.PUT_LINE(i || '. address is ' || emp_address(i));
```

```
END LOOP;
```

```
DBMS_OUTPUT.PUT_LINE('Modify emp_address VARRAY ');
```

```
emp_address(1) := 'Sydeny, AUS';
```

```
DBMS_OUTPUT.PUT_LINE('Accessing VARRAY based on index,modified address  
is '
```

```
||emp_address(1)); -- notice modified value here.
```

```
--emp_address.DELETE(2);--Delete operation on VARRAY is not allowed.
```

```
END;
```

OUTPUT:

VARRAY elements count is 3

Address display - Iteration over VARRAY

1. address is HYD,IND

2. address is NY,USA

3. address is BANG,IND

Modify emp_address VARRAY

Accessing VARRAY based on index, modified address is Sydeny, AUS

NESTED TABLES:

It is a table (with rows and columns) that is stored in database table as data of a column in no particular order. When that table is retrieved from database in PL/SQL context, PL/SQL indexes all rows starting from 1 and based on index we can access each row of nested table using method:

```
nested_table_var(index).
```

Following diagram shows how Nested tables is stored in database table. Highlighted inner table in CUSTOMER_DETAILS column refers to Nested table type and stored as part of column data.



Nested table creation and its initialization

Syntax of Nested table creation is as follows, (nested_table_type_def with collection) :

TYPE type IS {TABLE OF datatype [NOT NULL] }

we have an Customer_detail_object is a Object TYPE and it stores customer details and nested table is collection of that object- each row of nested table is customer_detail_object. If you do not understand what is this Object, do not worry we will revisit it again, for the time being just assume it is a container which can store different data types. Follow following steps and execute query in sequence :

Step 1: Create Object type having fields CustID, cust_name, cust_address, execute below query to create Object named Customer_detail_object.

--create Object Customer_detail_object : Created in schema level.

```
create type Customer_detail_object as object  
(  
    custID NUMBER(14),  
    cust_name varchar2(25),  
    cust_address varchar2(100)  
);
```



Step2: Now nested table type CUSTOMER_DETAILS of object type

Customer_detail_object.888888

--Create TABLE of object Customer_detail_object: Created in schema level

create type CUSTOMER_DETAILS as Table of Customer_detail_object;

Step 3: Create a table, PRODUCTS_CUSTOMRS_DETAILS, in database with a fields of type CUSTOMER_DETAILS (while creating table we specify about CUSTOMER_DETAILS as nested table).

--create table in database , NESTED TABLE clause is mandatory to append

create table PRODUCTS_CUSTOMRS_DETAILS

(

product_id number(5),

product_name varchar2(30),

CUSTOMER_DETAILS HR.CUSTOMER_DETAILS

) NESTED **TABLE** CUSTOMER_DETAILS STORE AS CUSTOMRS_OBJECTS;

Step 4: Insert rows in table. We have created two rows and each row has CUSTOMER_DETAILS table with two rows. If constructor used is empty, nested table will be empty not NULL.

Insert data into table

insert into PRODUCTS_CUSTOMRS_DETAILS

values(1,'P1',

CUSTOMER_DETAILS(

Customer_detail_object(1,'RSQ','BANG,INDIA'),

Customer_detail_object(2,'RTA','AUSTIN,USA')

));



```
insert into PRODUCTS_CUSTOMRS_DETAILS values (2,'P2',  
CUSTOMER_DETAILS (  
    Customer_detail_object (1,'RSQ','BANG,INDIA'),  
    Customer_detail_object(2,'BAC','NY,USA')  
));  
commit;
```

Now we have completed set-up to query database and see the stored result from PL/SQL program.

declare

```
customerDetails_Tab CUSTOMER_DETAILS;
```

begin

--insert a record in database table with nested table data

```
insert into products_CUSTOMRS_DETAILS
```

```
values(3,'P3',
```

```
CUSTOMER_DETAILS(  
    Customer_detail_object(1,'ACV','HYD,INDIA'),  
    Customer_detail_object(2,'ERT','AUSTIN,USA')  
));
```

```
));
```

```
commit;
```

--select record and store nested table value in customerDetails_Tab

```
select CUSTOMER_DETAILS into customerDetails_Tab
```

```
from products_CUSTOMRS_DETAILS
```

```
where product_id = 1;
```

--update nested table column in database

```
update products_CUSTOMRS_DETAILS set CUSTOMER_DETAILS =
```

```
customerDetails_Tab
```

```
where product_id = 3;
```

```
commit;
```



end;

Here we played around with DML statements and treating inner table as atomic value (Insert, select or update nested table in column). We can deal with individual row of nested table using TABLE command as follows:

```
select * from table ( select CUSTOMER_DETAILS from  
products_CUSTOMRS_DETAILS where product_id = 1);
```

Above query executes and it displays nested tables corresponding to row with product_id = 1, as follows :



PL/SQL records:

Use PL/SQL records when you want to store values of different data types but only one occurrence at a time. PL/SQL record must contain one or more components (called fields) of any scalar, RECORD or Index by table data type.

Creating PL/SQL records:

Syntax:

```
TYPE type_name is RECORD  
  (field_declaration [, field_declaration ]...);  
identifier type name;
```

field_declaration:

```
field_name {field_type | variable%type  
  | table.column%type | table%rowtype}  
  [[NOT NULL] {:= | DEFAULT} expr]
```

Example:



```
TYPE emp_record_type IS RECORD
(lname VARCHAR2(20),
 job_id VARCHAR2(30) DEFAULT 'Developer',
 salary NUMBER emp.salary%TYPE);
emp_type emp_record_type;
...
```

Using %ROWTYPE in composite data type: The %ROWTYPE attribute is useful when the number and data types of the underlying database columns is unknown and retrieving a row with the SELECT * from statement.

Example:

```
DECLARE
emp_rec employees%ROWTYPE;
BEGIN
SELECT * INTO emp_rec FROM employees
WHERE employee_id = 124;
emp_rec.hire_date := SYSDATE;
UPDATE EMP SET ROW = emp_rec;
END;
```

8.3.3. REFERENCE DATA TYPES:

In PL/SQL, **REF** data types are pointers that uniquely identify a piece of data as an object. A reference can be established between an existent valid object and a table or type attribute using the **REF** pointer data type. An attribute referring to a nonexistent object leads to "dangling" situation. Note that a NULL object reference is different from a Dangling Reference.

To insert data into a ref column, the REF function is used to get an object instance reference.

Syntax:

```
[ATTRIBUTE | COLUMN] REF [OBJECT TYPE]
```




Example:

The example code below shows the declaration of the REF attribute in table REFTAB. It points to TYPE_REFT object instance. Note the usage of REF as a function and the reference to an object instance.

```
CREATE OR REPLACE TYPE TYP_REFT AS OBJECT
```

```
(A NUMBER,
```

```
B NUMBER);
```

```
/
```

```
Type created.
```

```
CREATE TABLE REFTT OF TYP_REFT;
```

```
Table created.
```

```
CREATE TABLE REFTAB
```

```
(ID REF TYP_REFT);
```

```
Table created.
```

```
INSERT INTO REFTT VALUES (1,2);
```

```
1 row created.
```

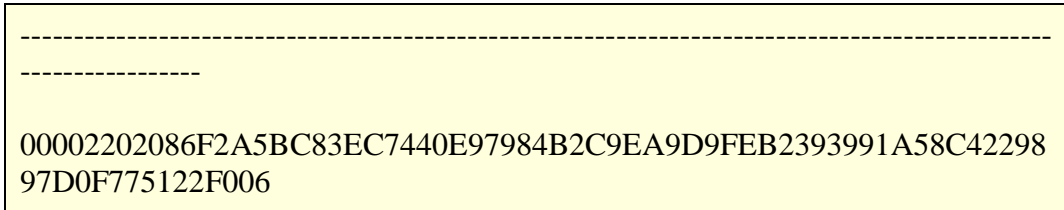
```
INSERT INTO REFTAB
```

```
SELECT REF(T) FROM REFTT T
```

```
1 row created.
```

```
SELECT * FROM REFTAB;
```

```
ID
```



8.3.4. LARGE OBJECT DATATYPES (LOB TYPES)

LOB data types use to store large object such as image, video, graphics, text, audio. Maximum size up to 4 Gigabytes. Following are LOB data types in SQL.

Datatype	Description	Storage(Maximum)
BFILE	BFILE data type to store a large binary object into Operating System file. This data type variable store full file locator's path, which points to a stored binary object within a server. BFILE data type read-only, you can't modify them.	Size: up to 4GB (2 ³² - 1 byte)
BLOB	BLOB data type same as BFILE data type to store an unstructured binary object into Operating System file. BLOB type fully supported transactions are recoverable and replicated.	Size: 8 TB to 128 TB (4GB - 1) * DB_BLOCK_SIZE
CLOB	CLOB data type to store large blocks of character data into Database. Store single byte and multi-byte character data. CLOB type fully supported transactions are recoverable and replicated.	Size: 8 TB to 128 TB (4GB - 1) * DB_BLOCK_SIZE



NCLOB	NCLOB data type to store large blocks of NCHAR data into Database. Store single byte and multi-byte character data. NCLOB type fully supported transactions are recoverable and replicated.	Size: 8 TB to 128 TB (4GB - 1) * DB_BLOCK_SIZE
-------	---	--

8.3.5. UNKNOWN COLUMN TYPES

PL/SQL this data type is used when column type is not know.

Datatype	Description
%Type	This data type is used to store value unknown data type column in a table. The column is identified by %type data type.
%RowType	This data type is used to store values unknown data type in all columns in a table. All columns are identified by %RowType datatype.
%RowID	RowID is data type. RowID is two types extended or restricted. Extended return 0 and restricted return 1 otherwise return the row number. Function of Row ID:

8.3.6. USER-DEFINED SUBTYPES

PL/SQL gives you the control to create your own sub data type that inherited from a predefined base type. Subtypes can increase reliability and provide compatibility with ANSI/ISO type. Several predefined subtypes are in a STANDARD package.

Defining Subtypes

The declarative part of PL/SQL block using the following syntax,



```
SUBTYPE subtype_name IS base_type[(constraint)] [NOT NULL];
```

Following the example, predefined data type inherits from CHARACTER and INTEGER data type to make a new sub type,

```
SUBTYPE CHARACTER IS CHAR;
```

```
SUBTYPE INTEGER IS NUMBER(10,4); -- allows for numbers
```

Example:

```
DECLARE
```

```
    SUBTYPE message IS varchar2(25);
```

```
    SUBTYPE age IS INTEGER(2,0);
```

```
    description message;
```

```
    ages age;
```

```
BEGIN
```

```
    description := 'Web Developer';
```

```
    ages := 22;
```

```
    dbms_output.put_line('I am ' || description || ' and I am ' || ages || ' years Old.');
```

```
END;
```

OUTPUT:

I am Web Developer and I am 22 years Old.

PL/SQL procedure successfully completed.

8.4. PL/SQL - VARIABLES

A variable is a meaningful name which facilitates a programmer to store data temporarily during the execution of code. It helps you to manipulate data in PL/SQL programs. It is nothing except a name given to a storage area.



Each variable in the PL/SQL has a specific data type which defines the size and layout of the variable's memory. A variable should not exceed 30 characters. Its letter optionally followed by more letters, dollar signs, numerals, underscore etc.

VARIABLE DECLARATION IN PL/SQL

PL/SQL variables must be declared in the declaration section or in a package as a global variable. When you declare a variable, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

SYNTAX FOR DECLARING VARIABLE

variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]

Here, variable_name is a valid identifier in PL/SQL and datatype must be valid PL/SQL data type. A data type with size, scale or precision limit is called a constrained declaration. The constrained declaration needs less memory than unconstrained declaration.

NAMING RULES FOR PL/SQL VARIABLES:

The variable in PL/SQL must follow some naming rules like other programming languages.

- The variable_name should not exceed 30 characters.
- The name of the variable must begin with ASCII letter. The PL/SQL is not case sensitive so it could be either lowercase or uppercase. For example: v_data and V_DATA refer to the same variables.
- You should make your variable easy to read and understand, after the first character, it may be any number, underscore (_) or dollar sign (\$).
- NOT NULL is an optional specification on the variable.

INITIALIZING VARIABLES IN PL/SQL

Every time you declare a variable, PL/SQL defines a default value NULL to it. If you want to initialize a variable with other value than NULL value, you can do so during the declaration, by using any one of the following methods.



- The DEFAULT keyword
 - The assignment operator
- ```
counter binary_integer := 0;
greetings varchar2(20) DEFAULT 'Hello Javaprogram';
```

It also specify NOT NULL constraint to avoid NULL value. If you specify the NOT NULL constraint, you must assign an initial value for that variable. You must have a good programming skill to initialize variable properly otherwise, sometimes program would produce unexpected result.

#### Example of initializing variable

1. DECLARE
2. a integer := 30;
3. b integer := 40;
4. c integer;
5. f real;
6. BEGIN
7. c := a + b;
8. dbms\_output.put\_line('Value of c: ' || c);
9. f := 100.0/3.0;
10. dbms\_output.put\_line('Value of f: ' || f);
11. END;

#### OUTPUT:

1. Value of c: 70
2. Value of f: 33.333333333333333333
- 3.
4. PL/SQL procedure successfully completed.



### Variable Scope in PL/SQL:

PL/SQL allows nesting of blocks. A program block can contain another inner block. If you declare a variable within an inner block, it is not accessible to an outer block.

There are two types of variable scope:

- Local Variable: Local variables are the inner block variables which are not accessible to outer blocks.
- Global Variable: Global variables are declared in outermost block.

### Example of Local and Global variables

```
1. DECLARE
2. -- Global variables
3. num1 number := 95;
4. num2 number := 85;
5. BEGIN
6. dbms_output.put_line('Outer Variable num1: ' || num1);
7. dbms_output.put_line('Outer Variable num2: ' || num2);
8. DECLARE
9. -- Local variables
10. num1 number := 195;
11. num2 number := 185;
12. BEGIN
13. dbms_output.put_line('Inner Variable num1: ' || num1);
14. dbms_output.put_line('Inner Variable num2: ' || num2);
15. END;
16. END;
17. /
```



### OUTPUT:

1. Outer Variable num1: 95
2. Outer Variable num2: 85
3. **Inner** Variable num1: 195
4. **Inner** Variable num2: 185
5. PL/SQL **procedure** successfully completed.

### 8.5. PL/SQL - OPERATORS

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulation. PL/SQL language is rich in built-in operators and provides the following types of operators :

1. Arithmetic operators
2. Relational operators
3. Comparison operators
4. Logical operators
5. String operators

#### 8.5.1. ARITHMETIC OPERATORS

Following table shows all the arithmetic operators supported by PL/SQL. Let us assume **variable A** holds 10 and **variable B** holds 5, then

| Operator | Description                                                       | Example                 |
|----------|-------------------------------------------------------------------|-------------------------|
| +        | Adds two operands                                                 | A + B will give 15      |
| -        | Subtracts second operand from the first                           | A - B will give 5       |
| *        | Multiplies both operands                                          | A * B will give 50      |
| /        | Divides numerator by de-numerator                                 | A / B will give 2       |
| **       | Exponentiation operator, raises one operand to the power of other | A ** B will give 100000 |





### 8.5.2. RELATIONAL OPERATORS

Relational operators compare two expressions or values and return a Boolean result. Following table shows all the relational operators supported by PL/SQL.

Let us assume **variable A** holds 10 and **variable B** holds 20, then

| Operator    | Description                                                                                                                     | Example               |
|-------------|---------------------------------------------------------------------------------------------------------------------------------|-----------------------|
| =           | Checks if the values of two operands are equal or not, if yes then condition becomes true.                                      | (A = B) is not true.  |
| != <><br>~= | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.                     | (A != B) is true.     |
| >           | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.             | (A > B) is not true.  |
| <           | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.                | (A < B) is true.      |
| >=          | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <=          | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.    | (A <= B) is true.     |

### 8.5.3. COMPARISON OPERATORS

Comparison operators are used for comparing one expression to another. The result is always either **TRUE**, **FALSE** or **NULL**.



| Operator | Description                                                                                                                                                 | Example                                                                                                                    |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| LIKE     | The LIKE operator compares a character, string, or CLOB value to a pattern and returns TRUE if the value matches the pattern and FALSE if it does not.      | If 'Zara Ali' like 'Z% A_i' returns a Boolean true, whereas, 'Nuha Ali' like 'Z% A_i' returns a Boolean false.             |
| BETWEEN  | The BETWEEN operator tests whether a value lies in a specified range. x BETWEEN a AND b means that $x \geq a$ and $x \leq b$ .                              | If $x = 10$ then, x between 5 and 20 returns true, x between 5 and 10 returns true, but x between 11 and 20 returns false. |
| IN       | The IN operator tests set membership. x IN (set) means that x is equal to any member of set.                                                                | If $x = 'm'$ then, x in ('a', 'b', 'c') returns Boolean false but x in ('m', 'n', 'o') returns Boolean true.               |
| IS NULL  | The IS NULL operator returns the BOOLEAN value TRUE if its operand is NULL or FALSE if it is not NULL. Comparisons involving NULL values always yield NULL. | If $x = 'm'$ , then 'x is null' returns Boolean false.                                                                     |

#### 8.5.4. LOGICAL OPERATORS

Following table shows the Logical operators supported by PL/SQL. All these operators work on Boolean operands and produce Boolean results. Let us assume **variable A** holds true and **variable B** holds false, then



| Operator | Description                                                                                                                                             | Examples               |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| and      | Called the logical AND operator. If both the operands are true then condition becomes true.                                                             | (A and B) is false.    |
| or       | Called the logical OR Operator. If any of the two operands is true then condition becomes true.                                                         | (A or B) is true.      |
| not      | Called the logical NOT Operator. Used to reverse the logical state of its operand. If a condition is true then Logical NOT operator will make it false. | not (A and B) is true. |

### 8.5.5. PL/SQL OPERATOR PRECEDENCE

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example,  $x = 7 + 3 * 2$ ; here,  $x$  is assigned **13**, not 20 because operator  $*$  has higher precedence than  $+$ , so it first gets multiplied with  $3*2$  and then adds into **7**.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

The precedence of operators goes as follows: =, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN.



| Operator   | Operation                            |
|------------|--------------------------------------|
| **         | Exponentiation                       |
| +, -       | identity, negation                   |
| *, /       | multiplication, division             |
| +, -,      | addition, subtraction, concatenation |
| Comparison |                                      |
| NOT        | logical negation                     |
| AND        | conjunction                          |
| OR         | inclusion                            |

### 8.6. PL/SQL - CONDITIONS

Decision-making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical conditional (i.e., decision making) structure found in most of the programming languages :



PL/SQL programming language provides following types of decision-making statements.  
Click the following links to check their detail.

| Statement & Description                                                                                                                                                                                                                                                                                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>IF - THEN statement</b><br>The <b>IF statement</b> associates a condition with a sequence of statements enclosed by the keywords <b>THEN</b> and <b>END IF</b> . If the condition is true, the statements get executed and if the condition is false or NULL then the IF statement does nothing.     |
| <b>IF-THEN-ELSE statement</b><br><b>IF statement</b> adds the keyword <b>ELSE</b> followed by an alternative sequence of statement. If the condition is false or NULL, then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed. |
| <b>IF-THEN-ELSIF statement</b><br>It allows you to choose between several alternatives.                                                                                                                                                                                                                 |
| <b>Case statement</b><br>Like the IF statement, the <b>CASE statement</b> selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than                                                                                                |



multiple Boolean expressions. A selector is an expression whose value is used to select one of several alternatives.

#### **Searched CASE statement**

The searched CASE statement **has no selector**, and its WHEN clauses contain search conditions that yield Boolean values.

#### **nested IF-THEN-ELSE**

You can use one **IF-THEN** or **IF-THEN-ELSIF** statement inside another **IF-THEN** or **IF-THEN-ELSIF** statement(s).

### **8.7. PL/SQL - LOOPS**



A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages –

PL/SQL provides the following types of loop to handle the looping requirements. Click the following links to check their detail.



| Loop Type & Description                                                                                                                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>PL/SQL Basic LOOP</b></p> <p>In this loop structure, sequence of statements is enclosed between the LOOP and the END LOOP statements. At each iteration, the sequence of statements is executed and then control resumes at the top of the loop.</p> |
| <p><b>PL/SQL WHILE LOOP</b></p> <p>Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.</p>                                                                                  |
| <p><b>PL/SQL FOR LOOP</b></p> <p>Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.</p>                                                                                                              |
| <p><b>Nested loops in PL/SQL</b></p> <p>You can use one or more loop inside any another basic loop, while, or for loop.</p>                                                                                                                                |

### LABELING A PL/SQL LOOP

PL/SQL loops can be labeled. The label should be enclosed by double angle brackets (<< and >>) and appear at the beginning of the LOOP statement. The label name can also appear at the end of the LOOP statement. You may use the label in the EXIT statement to exit from the loop.

```
DECLARE
 i number(1);
 j number(1);
BEGIN
 << outer_loop >>
```



```
FOR i IN 1..3 LOOP
 << inner_loop >>
 FOR j IN 1..3 LOOP
 dbms_output.put_line('i is: ' || i || ' and j is: ' || j);
 END loop inner_loop;
END loop outer_loop;
END; /
```

#### OUTPUT:

i is: 1 and j is: 1

i is: 1 and j is: 2

i is: 1 and j is: 3

i is: 2 and j is: 1

i is: 2 and j is: 2

i is: 2 and j is: 3

i is: 3 and j is: 1

i is: 3 and j is: 2

i is: 3 and j is: 3

PL/SQL procedure successfully completed.

#### THE LOOP CONTROL STATEMENTS

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. PL/SQL supports the following control statements.

Labeling loops also help in taking the control outside a loop. Click the following links to check their details.





| Control Statement & Description                                                                                                               |
|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <p>EXIT statement</p> <p>The Exit statement completes the loop and control passes to the statement immediately after the END LOOP.</p>        |
| <p>CONTINUE statement</p> <p>Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.</p> |
| <p>GOTO statement</p> <p>Transfers control to the labeled statement. Though it is not advised to use the GOTO statement in your program.</p>  |

### 8.8. PL/SQL – STRINGS

The string in PL/SQL is actually a sequence of characters with an optional size specification. The characters could be numeric, letters, blank, special characters or a combination of all. PL/SQL offers three kinds of strings :

- **Fixed-length strings** – In such strings, programmers specify the length while declaring the string. The string is right-padded with spaces to the length so specified.
- **Variable-length strings** – In such strings, a maximum length up to 32,767, for the string is specified and no padding takes place.
- **Character large objects (CLOBs)** – These are variable-length strings that can be up to 128 terabytes.

PL/SQL strings could be either variables or literals. A string literal is enclosed within quotation marks.

For example:



### 'This is a string literal.' Or 'hello world'

To include a single quote inside a string literal, you need to type two single quotes next to one another. For example:

'this isn't what it looks like'

#### 8.8.1. DECLARING STRING VARIABLES

Oracle database provides numerous string datatypes, such as CHAR, NCHAR, VARCHAR2, NVARCHAR2, CLOB, and NCLOB. The datatypes prefixed with an 'N' are '**national character set**' datatypes, that store Unicode character data.

If you need to declare a variable-length string, you must provide the maximum length of that string.

For example, the VARCHAR2 data type. The following example illustrates declaring and using some string variables:

```
DECLARE
 name varchar2(20);
 company varchar2(30);
 introduction clob;
 choice char(1);
BEGIN
 name := 'John Smith';
 company := 'Infotech';
 introduction := ' Hello! I'm John Smith from Infotech.';
 choice := 'y';
 IF choice = 'y' THEN
 dbms_output.put_line(name);
 dbms_output.put_line(company);
 dbms_output.put_line(introduction);
```



```
END IF;
END; /
```

**OUTPUT:**

```
John Smith
Infotech
Hello! I'm John Smith from Infotech.
PL/SQL procedure successfully completed
```

To declare a fixed-length string, use the CHAR datatype. Here you do not have to specify a maximum length for a fixed-length variable. If you leave off the length constraint, Oracle Database automatically uses a maximum length required.

The following two declarations are identical –

```
red_flag CHAR(1) := 'Y';
red_flag CHAR := 'Y';
```

**8.8.2. PL/SQL STRING FUNCTIONS AND OPERATORS**

PL/SQL offers the concatenation operator (||) for joining two strings.

The following table provides the string functions provided by PL/SQL:

| Function & Purpose                                                                                     |
|--------------------------------------------------------------------------------------------------------|
| <b>ASCII(x):</b> Returns the ASCII value of the character x.                                           |
| <b>CHR(x):</b> Returns the character with the ASCII value of x.                                        |
| <b>CONCAT(x, y):</b> Concatenates the strings x and y and returns the appended string.                 |
| <b>INITCAP(x):</b> Converts the initial letter of each word in x to uppercase and returns that string. |



**INSTR(x, find\_string [, start] [, occurrence]):** Searches for **find\_string** in **x** and returns the position at which it occurs.

**LENGTH(x):** Returns the number of characters in **x**.

**LOWER(x):** Converts the letters in **x** to lowercase and returns that string.

**SUBSTR(x, start [, length]):** Returns a substring of **x** that begins at the position specified by **start**. An optional length for the substring may be supplied.

**TRIM([trim\_char FROM] x):** Trims characters from the left and right of **x**.

**UPPER(x):** Converts the letters in **x** to uppercase and returns that string.

DECLARE

```
greetings varchar2(30) := '.....Hello World.....';
```

BEGIN

```
dbms_output.put_line(RTRIM(greetings,'.');
```

```
dbms_output.put_line(LTRIM(greetings, '.'));
```

```
dbms_output.put_line(TRIM('.' from greetings));
```

END;

/

**OUTPUT:**

.....Hello World

Hello World.....

Hello World

PL/SQL procedure successfully completed.

### 8.9. PL/SQL – ARRAYS

The PL/SQL programming language provides a data structure called the **VARRAY**, which can store a fixed-size sequential collection of elements of the same type. A varray is used to store an ordered collection of data, however it is often better to think of an array as a collection of variables of the same type.



All varrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



An array is a part of collection type data and it stands for variable-size arrays. We will study other collection types in a later chapter '**PL/SQL Collections**'.

Each element in a **varray** has an index associated with it. It also has a maximum size that can be changed dynamically.

### Creating a Varray Type

A varray type is created with the **CREATE TYPE** statement. You must specify the maximum size and the type of elements stored in the varray.

**The basic syntax for creating a VARRAY type at the schema level is :**

```
CREATE OR REPLACE TYPE varray_type_name IS VARRAY(n) of <element_type>
```

Where,

- *varray\_type\_name* is a valid attribute name,
- *n* is the number of elements (maximum) in the varray,
- *element\_type* is the data type of the elements of the array.

Maximum size of a varray can be changed using the **ALTER TYPE** statement.



**For Example:**

```
CREATE Or REPLACE TYPE namearray AS VARRAY(3) OF VARCHAR2(10);
/ Type created.
```

The basic syntax for creating a VARRAY type within a PL/SQL block is :

```
TYPE varray_type_name IS VARRAY(n) OF <element_type>
```

**For Example:**

```
TYPE namearray IS VARRAY(5) OF VARCHAR2(10);
Type grades IS VARRAY(5) OF INTEGER;
```

Let us now work out on a few examples to understand the concept :

**Example 1:**

The following program illustrates the use of varrays :

```
DECLARE
 type namesarray IS VARRAY(5) OF VARCHAR2(10);
 type grades IS VARRAY(5) OF INTEGER;
 names namesarray;
 marks grades;
 total integer;
BEGIN
 names := namesarray('Kavita', 'Pritam', 'Ayan', 'Rishav', 'Aziz');
 marks:= grades(98, 97, 78, 87, 92);
 total := names.count;
 dbms_output.put_line('Total '|| total || ' Students');
 FOR i in 1 .. total LOOP
```



```
dbms_output.put_line('Student: ' || names(i) || '
Marks: ' || marks(i));
END LOOP;
END;
/
```

### OUTPUT:

Total 5 Students

Student: Kavita Marks: 98

Student: Pritam Marks: 97

Student: Ayan Marks: 78

Student: Rishav Marks: 87

Student: Aziz Marks: 92

PL/SQL procedure successfully completed.

### SAMPLE PROGRAMS:

#### Example 1: Simple display statement

##### Step 1:

```
SQL> Set Serveroutput On;
```

##### Step 2:

```
SQL> begin
```

```
2 dbms_output.put_line('Welcome to PL/SQL');
```

```
3 end;
```

```
4 /
```

##### Output:

Welcome to PL/SQL

PL/SQL procedure successfully completed.



### Example 2: Calculation using Variables

#### Step 1:

```
SQL> Set Serveroutput On;
```

#### Step 2:

```
SQL>declare
2 a number;
3 b number;
4 c number;
5 begin
6 a :=10;
7 b :=20;
8 c :=a+b;
9 dbms_output.put_line('A + B = '||c);
10 end;
11 /
```

#### Output:

A + B = 30

PL/SQL procedure successfully completed.

### Example 3: Using Table:

#### Step1 :

```
SQL> create table stu(rno number(3), name varchar2(20), DBMS number(3), DS
number(3), CO number(3), total number(4), average number(4), result varchar2(8));
```

Table created.

#### Step 2:

```
SQL>insert into stuvalues(&rno,'&name',&dbms,&ds,&co,&total,&average,&result);
```

#### Step 3:

```
SQL> select * from stu;
```





| RNO | NAME  | DBMS | DS | CO | TOTAL | AVERAGE | RESULT |
|-----|-------|------|----|----|-------|---------|--------|
| 1   | shree | 90   | 90 | 90 | 270   | 90      | pass   |
| 2   | priya | 80   | 80 | 80 | 240   | 80      | pass   |

**Step 4:** set serveroutput on;

**Step 5:** Go to Edit mode and type the following coding.

```
declare
no stu.rno%type;
name stu.name%type;
m1 stu.dbms%type;
m2 stu.ds%type;
m3 stu.co%type;(or) st stu%rowtype;
tot stu.total%type;
aveg stu.average%type;
res stu.result%type;
begin
no :=&no;
select rno,name,dbms,ds,co,total,average,result into
no,name,m1,m2,m3,tot,aveg,res from stu where rno=no;
dbms_output.new_line();
dbms_output.put_line('*****');
dbms_output.put_line('Student Marks Statement');
dbms_output.put_line('*****');
dbms_output.new_line();
dbms_output.put_line('Register Number : ||no);
dbms_output.put_line('Student Name : ||name);
dbms_output.put_line('DBMS Mark : ||m1);
dbms_output.put_line('DS Mark : ||m2);
dbms_output.put_line('CO Mark : ||m3);
```



```
dbms_output.put_line('Total Marks : ||tot);
dbms_output.put_line('Average : ||aveg);
dbms_output.put_line('Result Pass/Fail : ||res);
end;
/
```

### Step 6:

```
SQL> /
```

Enter value for no: 2

### Output:

```
old 11: no :=&no;
```

```
new 11: no :=2;
```

```

```

```
Student Marks Statement
```

```

```

```
Register Number : 2
```

```
Student Name : priya
```

```
DBMS Mark : 80
```

```
DS Mark : 80
```

```
CO Mark : 80
```

```
Total Marks : 240
```

```
Average : 80
```

```
Result Pass/Fail : pass
```

PL/SQL procedure successfully completed.

## 8.10. PL/SQL TABLES AND RECORDS

**PL/SQL tables** use a primary key to give you array-like access to rows. The number of rows in a PL/SQL table can increase dynamically. The **PL/SQL tables** grows as new rows are added. **PL/SQL tables** can have one column and a primary key, neither of which can



be named. The column can belong to any scalar type, but the primary key must belong to type BINARY\_INTEGER.

A **PL/SQL tables** can consist of one simple datatype or be defined as a type of record and is sometimes referred to as an **Indexbytable**. Rows in a PL/SQL table do not have to be contiguous. Objects of type TABLE are known as **PL/SQL tables**.

PL/SQL has two composite data types:

1. TABLE
2. RECORD

### 8.10.1. TABLES

**Tables with simple datatypes can be populated as:**

```
<variable>(<integer>) := <value>;
```

**Tables with complex datatypes will need the columns populated individually as:**

```
<variable>(<integer>).<column_name> := <value>;
```

**Or from a cursor:**

```
fetch <cursor_name> into <variable>(<integer>);
```

#### **Example1 of PL/SQL Table:**

Type my\_first\_table is table of varchar2(10)

Index by binary\_integer;

Var\_of\_table my\_first\_table;



```
Var_of_table(1) := 'hello world';
```

```
Var_of_table(2) := 'bye';
```

**To delete individual records from PL/SQL tables:**

```
Var_of_emp .delete(1);
```

**To remove all entries from a PL/SQL table:**

```
Var_of_emp.delete;
```

Or

```
Var_of_emp := var1_of_emp
```

Where var1\_of\_emp does not contain any value, it is empty.

**COUNT method can be used to return number of records in a PL/SQL Table.**

```
Var_of_table.count
```

**First, Next and Last methods of PL/SQL Tables.**

First is for first index in the **PL/SQL Tables**.

Last is for last index in the **PL/SQL Tables**.

Next is for next index in the **PL/SQL Tables**.

**Example showing First and Next method of PL/SQL tables**

```
SQL> set serveroutput on
```

```
SQL> Declare
```

```
2
```

```
3 Type my_dept_table is table of varchar2(20)
```

```
4 Index by binary_integer;
```

```
5
```



```
6 Var_of_dept my_dept_table;
7 Index_value number;
8
9 Begin
10
11 For dept_rec in (select * from dept) loop
12 Var_of_dept(dept_rec.deptno) := dept_rec.dname;
13 End loop;
14
15 Index_value := var_of_dept.first;
16 Loop
17 Exit when index_value is null;
18 Dbms_output.put_line (index_value || ' ' || var_of_dept(index_value));
19 Index_value := var_of_dept.next(index_value);
20 End loop;
21 End;
22 /
```

### **OUTPUT**

```
10 ACCOUNTING
20 RESEARCH
30 SALES
40 OPERATIONS
```

PL/SQL procedure successfully completed.



## 8.10.2. RECORDS

A **record** is a data structure that can hold data items of different kinds. Records consist of different fields, similar to a row of a database table. For example, you want to keep track of your books in a library.

To track the following attributes about each book, such as Title, Author, Subject, Book ID. A record containing a field for each of these items allows treating a BOOK as a logical unit and allows you to organize and represent its information in a better way.

PL/SQL can handle the following types of records:

- a. Table-based records
- b. Cursor-based records
- c. User-defined records

### a. TABLE-BASED RECORDS

The %ROWTYPE attribute enables a programmer to create **tablebased** and **cursorbased** records.

The following example illustrates the concept of **table-based** records. We will be using the CUSTOMERS table we had created:

```
DECLARE
 customer_rec customers%rowtype;
BEGIN
 SELECT * into customer_rec
 FROM customers
 WHERE id = 5;
 dbms_output.put_line('Customer ID: ' || customer_rec.id);
 dbms_output.put_line('Customer Name: ' || customer_rec.name);
```



```
dbms_output.put_line('Customer Address: ' || customer_rec.address);
dbms_output.put_line('Customer Salary: ' || customer_rec.salary);
END;
/
```

#### **OUTPUT:**

Customer ID: 5

Customer Name: Hardik

Customer Address: Bhopal

Customer Salary: 9000

PL/SQL procedure successfully completed.

#### **b. CURSOR-BASED RECORDS**

The following example illustrates the concept of **cursor-based** records. We will be using the CUSTOMERS table we had created:

```
DECLARE
 CURSOR customer_cur is
 SELECT id, name, address
 FROM customers;
 customer_rec customer_cur%rowtype;
BEGIN
 OPEN customer_cur;
 LOOP
 FETCH customer_cur into customer_rec;
 EXIT WHEN customer_cur%notfound;
 DBMS_OUTPUT.put_line(customer_rec.id || ' ' || customer_rec.name);
```



```
END LOOP;
END;
/
```

**OUTPUT:**

```
1 Ramesh
2 Khilan
3 kaushik
4 Chaitali
5 Hardik
6 Komal
PL/SQL procedure successfully completed.
```

**c. USER-DEFINED RECORDS**

PL/SQL provides a user-defined record type that allows you to define the different record structures. These records consist of different fields. Suppose you want to keep track of your books in a library.

If we want to track the following attributes about each book :

- Title
- Author
- Subject
- Book ID

**DEFINING A RECORD**

The record type is defined as :

TYPE

type\_name IS RECORD

( field\_name1 datatype1 [NOT NULL] [:= DEFAULT EXPRESSION],





```
field_name2 datatype2 [NOT NULL] [:= DEFAULT EXPRESSION],
...
field_nameN datatypeN [NOT NULL] [:= DEFAULT EXPRESSION];
record-name type_name;
```

The Book record is declared in the following way :

```
DECLARE
TYPE books IS RECORD
(title varchar(50),
 author varchar(50),
 subject varchar(100),
 book_id number);
book1 books;
book2 books;
```

### ACCESSING FIELDS

To access any field of a record, we use the dot (.) operator. The member access operator is coded as a period between the record variable name and the field that we wish to access.

Following is an example to explain the usage of record :

```
DECLARE
type books is record
(title varchar(50),
 author varchar(50),
 subject varchar(100),
 book_id number);
book1 books;
```



```
book2 books;
BEGIN
-- Book 1 specification
book1.title := 'C Programming';
book1.author := 'Nuha Ali ';
book1.subject := 'C Programming Tutorial';
book1.book_id := 6495407;
-- Book 2 specification
book2.title := 'Telecom Billing';
book2.author := 'Zara Ali';
book2.subject := 'Telecom Billing Tutorial';
book2.book_id := 6495700;
-- Print book 1 record
dbms_output.put_line('Book 1 title : ' || book1.title);
dbms_output.put_line('Book 1 author : ' || book1.author);
dbms_output.put_line('Book 1 subject : ' || book1.subject);
dbms_output.put_line('Book 1 book_id : ' || book1.book_id);
-- Print book 2 record
dbms_output.put_line('Book 2 title : ' || book2.title);
dbms_output.put_line('Book 2 author : ' || book2.author);
dbms_output.put_line('Book 2 subject : ' || book2.subject);
dbms_output.put_line('Book 2 book_id : ' || book2.book_id);
END;
/
```

**OUTPUT:**

Book 1 title : C Programming

Book 1 author : Nuha Ali

Book 1 subject : C Programming Tutorial



Book 1 book\_id : 6495407

Book 2 title : Telecom Billing

Book 2 author : Zara Ali

Book 2 subject : Telecom Billing Tutorial

Book 2 book\_id : 6495700

PL/SQL procedure successfully completed.

### **8.11. PL/SQL CURSOR**

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement.

In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it. A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time.

There are two types of cursors:

- Implicit Cursors
- Explicit Cursors

#### **8.11.1. PL/SQL IMPLICIT CURSORS**

The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement. These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.

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

**For Example:**



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

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

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

**For Example:**

**Create customers table and have records:**

| ID | NAME   | AGE | ADDRESS   | SALARY |
|----|--------|-----|-----------|--------|
| 1  | Ramesh | 23  | Allahabad | 20000  |
| 2  | Suresh | 22  | Kanpur    | 22000  |



|   |         |    |           |       |
|---|---------|----|-----------|-------|
| 3 | Mahesh  | 24 | Ghaziabad | 24000 |
| 4 | Chandan | 25 | Noida     | 26000 |
| 5 | Alex    | 21 | Paris     | 28000 |
| 6 | Sunita  | 20 | Delhi     | 30000 |

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

**Create procedure:**

1. **DECLARE**
2. total\_rows number(2);
3. **BEGIN**
4. **UPDATE** customers
5. **SET** salary = salary + 5000;
6. **IF** sql%notfound **THEN**
7. dbms\_output.put\_line('no customers updated');
8. **ELSIF** sql% found **THEN**
9. total\_rows := sql%rowcount;
10. dbms\_output.put\_line( total\_rows || ' customers updated ');
11. **END IF;**
12. **END;**
13. /

**OUTPUT:**

6 customers updated  
PL/SQL procedure successfully completed.



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

**select \* from** customers;

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

### 8.11.2. PL/SQL EXPLICIT CURSORS

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

Syntax of explicit cursor

**CURSOR** cursor\_name **IS** select\_statement;

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

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



1) **Declare the cursor:** It defines the cursor with a name and the associated SELECT statement.

**Syntax for explicit cursor declaration**

**CURSOR name IS SELECT statement;**

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

**Syntax for cursor open: OPEN cursor\_name;**

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

**Syntax for cursor fetch: FETCH cursor\_name INTO variable\_list;**

4) **Close the cursor:** It is used to release the allocated memory. The following syntax is used to close the above-opened cursors.

**Syntax for cursor close: Close cursor\_name;**

**For Example**

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

**Create customers table and have records:**

| ID | NAME   | AGE | ADDRESS   | SALARY |
|----|--------|-----|-----------|--------|
| 1  | Ramesh | 23  | Allahabad | 20000  |
| 2  | Suresh | 22  | Kanpur    | 22000  |
| 3  | Mahesh | 24  | Ghaziabad | 24000  |



|   |         |    |       |       |
|---|---------|----|-------|-------|
| 4 | Chandan | 25 | Noida | 26000 |
| 5 | Alex    | 21 | Paris | 28000 |
| 6 | Sunita  | 20 | Delhi | 30000 |

**Create procedure:**

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

1. **DECLARE**
2.     c\_id customers.id%type;
3.     c\_name customers.name%type;
4.     c\_addr customers.address%type;
5.     **CURSOR** c\_customers **is**
6.         **SELECT** id, name, address **FROM** customers;
7. **BEGIN**
8.     **OPEN** c\_customers;
9.     **LOOP**
10.         **FETCH** c\_customers **into** c\_id, c\_name, c\_addr;
11.         **EXIT WHEN** c\_customers%notfound;
12.         dbms\_output.put\_line(c\_id || ' ' || c\_name || ' ' || c\_addr);
13.     **END LOOP;**
14.     **CLOSE** c\_customers;
15. **END;**
16.     /

**OUTPUT:**

- 1 Ramesh Allahabad
- 2 Suresh Kanpur
- 3 Mahesh Ghaziabad





4 Chandan Noida

5 Alex Paris

6 Sunita Delhi

PL/SQL procedure successfully completed.

### FOR EXAMPLE CURSOR PROGRAM

SQL> declare

2 cursor stucur is select \* from stu;

3 st stucur%rowtype;

4 begin

5 open stucur;

6 loop

7 fetch stucur into st;

8 exit when stucur%notfound;

9 st.total := st.dbms+st.ds+st.co;

10 st.average := st.total/3;

11 if( st.dbms >=50 and st.ds >=50 and st.co >=50) then

12 st.result := 'PASS';

13 else

14 st.result := 'FAIL';

15 end if;

16 update stu set total=st.total, average=st.average, result=st.result where rno=st.rno;

17 end loop;

18 end;

19 /

### Output:

PL/SQL procedure successfully completed.



**Step 4:**

SQL> select \* from stu;

| RNO | NAME  | DBMS | DS | CO | TOTAL | AVERAGE | RESULT |
|-----|-------|------|----|----|-------|---------|--------|
| 1   | shree | 90   | 90 | 90 | 270   | 90      | PASS   |
| 2   | priya | 80   | 80 | 80 | 240   | 80      | PASS   |
| 3   | anu   | 20   | 30 | 56 | 106   | 35      | FAIL   |
| 4   | suja  | 67   | 34 | 56 | 157   | 52      | FAIL   |

**8.12. PL/SQL EXCEPTION HANDLING**

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

There are two type of exceptions:

- System-defined Exceptions
- User-defined Exceptions

**8.12.1. PL/SQL SYSTEM-DEFINED EXCEPTIONS**

**Syntax for exception handling:**

Following is a general syntax for exception handling:

1. **DECLARE**
2. <declarations **section**>
3. **BEGIN**
4. <executable command(s)>



5. EXCEPTION
6. <exception handling goes here >
7. **WHEN** exception1 **THEN**
8.     exception1-handling-statements
9. **WHEN** exception2 **THEN**
10.    exception2-handling-statements
11. **WHEN** exception3 **THEN**
12.    exception3-handling-statements
13.    .....
14. **WHEN** others **THEN**
15.    exception3-handling-statements
16.    **END;**

### **PL/SQL PRE-DEFINED EXCEPTIONS**

There are many pre-defined exception in PL/SQL which are executed when any database rule is violated by the programs.

**For example:** NO\_DATA\_FOUND is a pre-defined exception which is raised when a SELECT INTO statement returns no rows.

Following is a list of some important pre-defined exceptions:

| <b>Exception</b> | <b>Oracle Error</b> | <b>SQL Code</b> | <b>Description</b>                                                                                                        |
|------------------|---------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| ACCESS_INTO_NULL | 06530               | -6530           | It is raised when a NULL object is automatically assigned a value.                                                        |
| CASE_NOT_FOUND   | 06592               | -6592           | It is raised when none of the choices in the "WHEN" clauses of a CASE statement is selected, and there is no else clause. |



|                    |       |       |                                                                                                                                                                                                                                     |
|--------------------|-------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| COLLECTION_IS_NULL | 06531 | -6531 | It is raised when a program attempts to apply collection methods other than exists to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray. |
| DUP_VAL_ON_INDEX   | 00001 | -1    | It is raised when duplicate values are attempted to be stored in a column with unique index.                                                                                                                                        |
| INVALID_CURSOR     | 01001 | -1001 | It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.                                                                                                             |
| INVALID_NUMBER     | 01722 | -1722 | It is raised when the conversion of a character string into a number fails because the string does not represent a valid number.                                                                                                    |
| LOGIN_DENIED       | 01017 | -1017 | It is raised when s program attempts to log on to the database with an invalid username or password.                                                                                                                                |
| NO_DATA_FOUND      | 01403 | +100  | It is raised when a select into statement returns no rows.                                                                                                                                                                          |
| NOT_LOGGED_ON      | 01012 | -1012 | It is raised when a database call is issued without being connected to the database.                                                                                                                                                |
| PROGRAM_ERROR      | 06501 | -6501 | It is raised when PL/SQL has an internal problem.                                                                                                                                                                                   |



|                  |       |        |                                                                                                        |
|------------------|-------|--------|--------------------------------------------------------------------------------------------------------|
| ROWTYPE_MISMATCH | 06504 | -6504  | It is raised when a cursor fetches value in a variable having incompatible data type.                  |
| SELF_IS_NULL     | 30625 | -30625 | It is raised when a member method is invoked, but the instance of the object type was not initialized. |
| STORAGE_ERROR    | 06500 | -6500  | It is raised when PL/SQL ran out of memory or memory was corrupted.                                    |
| TOO_MANY_ROWS    | 01422 | -1422  | It is raised when a SELECT INTO statement returns more than one row.                                   |
| VALUE_ERROR      | 06502 | -6502  | It is raised when an arithmetic, conversion, truncation, or size-constraint error occurs.              |
| ZERO_DIVIDE      | 01476 | 1476   | It is raised when an attempt is made to divide a number by zero                                        |

**Example of exception handling:**

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

SELECT\* FROM COUSTOMERS;

| ID | NAME   | AGE | ADDRESS   | SALARY |
|----|--------|-----|-----------|--------|
| 1  | Ramesh | 23  | Allahabad | 20000  |
| 2  | Suresh | 22  | Kanpur    | 22000  |



|   |         |    |           |       |
|---|---------|----|-----------|-------|
| 3 | Mahesh  | 24 | Ghaziabad | 24000 |
| 4 | Chandan | 25 | Noida     | 26000 |
| 5 | Alex    | 21 | Paris     | 28000 |
| 6 | Sunita  | 20 | Delhi     | 30000 |

1. **DECLARE**
2.     c\_id customers.id%type := 8;
3.     c\_name customers.name%type;
4.     c\_addr customers.address%type;
5. **BEGIN**
6.     **SELECT** name, address **INTO** c\_name, c\_addr
7.     **FROM** customers
8.     **WHERE** id = c\_id;
9.     DBMS\_OUTPUT.PUT\_LINE ('Name: ' || c\_name);
10.    DBMS\_OUTPUT.PUT\_LINE ('Address: ' || c\_addr);
11. **EXCEPTION**
12.    **WHEN** no\_data\_found **THEN**
13.    dbms\_output.put\_line('No such customer!');
14.    **WHEN** others **THEN**
15.    dbms\_output.put\_line('Error!');
16. **END;**
17. /

**OUTPUT:**

No such customer!

PL/SQL procedure successfully completed.



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

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

```
1. DECLARE
2. c_id customers.id%type := 5;
3. c_name customers.name%type;
4. c_addr customers.address%type;
5. BEGIN
6. SELECT name, address INTO c_name, c_addr
7. FROM customers
8. WHERE id = c_id;
9. DBMS_OUTPUT.PUT_LINE ('Name: ' || c_name);
10. DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
11. EXCEPTION
12. WHEN no_data_found THEN
13. dbms_output.put_line('No such customer!');
14. WHEN others THEN
15. dbms_output.put_line('Error!');
16. END;
17. /
```

**OUTPUT:**

Name: alex

Address: paris

PL/SQL procedure successfully completed.



### 8.12.2. PL/SQL USER-DEFINED EXCEPTIONS

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

#### Syntax for user define exceptions

1. **DECLARE**
2. my-exception EXCEPTION;

#### RAISING EXCEPTIONS

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

#### Syntax for raising an exception:

1. **DECLARE**
2. exception\_name EXCEPTION;
3. **BEGIN**
4. IF condition **THEN**
5. RAISE exception\_name;
6. **END IF;**
7. **EXCEPTION**
8. **WHEN** exception\_name **THEN**
9. statement;
10. **END;**





### 8.13. PL/SQL SUB PROGRAMS

A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'.

A subprogram can be invoked by another subprogram or program which is called the **calling program**. PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters.

PL/SQL provides two kinds of subprograms :

- **Functions** – These subprograms return a single value; mainly used to compute and return a value.
- **Procedures** – These subprograms do not return a value directly; mainly used to perform an action.

#### 8.13.1 PL/SQL FUNCTIONS

A function is same as a procedure except that it returns a value. Therefore, all the discussions of the previous chapter are true for functions too.

**Syntax to create a function:**

1. **CREATE** [OR REPLACE] **FUNCTION** function\_name [parameters]
2. [(parameter\_name [IN | OUT | IN OUT] type [, ...])]
3. **RETURN** return\_datatype
4. {**IS** | **AS**}
5. **BEGIN**
6. < function\_body >
7. **END** [function\_name];

**Here:**

- **Function\_name:** specifies the name of the function.



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

**The function must contain a return statement.**

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

**PL/SQL Function Example**

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

1. **create** or replace **function** adder(n1 in number, n2 in number)
2. **return** number
3. **is**
4. n3 number(8);
5. **begin**
6. n3 :=n1+n2;
7. **return** n3;
8. **end;**
9. /

**Calling PL/SQL Function:**

While creating a function, you have to give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. Once the function is called, the program control is transferred to the called function.

After the successful completion of the defined task, the call function returns program control back to the main program. To call a function you have to pass the required



parameters along with function name and if function returns a value then you can store returned value.

Following program calls the function totalCustomers from an anonymous block:

1. **DECLARE**
2.     c number(2);
3. **BEGIN**
4.     c := totalCustomers();
5.     dbms\_output.put\_line('Total no. of Customers: ' || c);
6. **END;**
7.     /

**OUTPUT:**

Total no. of Customers: 4

PL/SQL procedure successfully completed.

**PL/SQL Recursive Function**

A program or a subprogram can call another subprogram. When a subprogram calls itself, it is called recursive call and the process is known as recursion.

**Example to calculate the factorial of a number**

Let's take an example to calculate the factorial of a number. This example calculates the factorial of a given number by calling itself recursively.

1. **DECLARE**
2.     num number;
3.     factorial number;
4. **FUNCTION** fact(x number)
5.     **RETURN** number
6. **IS**
7.     f number;



```
8. BEGIN
9. IF x=0 THEN
10. f := 1;
11. ELSE
12. f := x * fact(x-1);
13. END IF;
14. RETURN f;
15. END;
16.
17. BEGIN
18. num:= 6;
19. factorial := fact(num);
20. dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
21. END;
22. /
```

#### **OUTPUT:**

Factorial 6 is 720

PL/SQL procedure successfully completed.

#### **PL/SQL Drop Function**

##### **Syntax for removing your created function:**

If you want to remove your created function from the database, you should use the following syntax.

```
1. DROP FUNCTION function_name;
```



### 8.13.2. PL/SQL PROCEDURE

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

The procedure contains a header and a body.

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

#### How to pass parameters in procedure:

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

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

#### PL/SQL CREATE PROCEDURE

##### Syntax for creating procedure

1. **CREATE [OR REPLACE] PROCEDURE** procedure\_name
2. [ (parameter [,parameter]) ]
3. **IS**
4. [declaration\_section]
5. **BEGIN**



6. executable\_section
7. [EXCEPTION
8. exception\_section]
9. **END** [procedure\_name];

### Create procedure example

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

### Table creation:

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

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

### Procedure Code:

1. **create** or replace **procedure** "INSERTUSER"
2. (id IN NUMBER,
3. **name** IN VARCHAR2)
4. **is**
5. **begin**
6. **insert into** user **values**(id,name);
7. **end;**
8. /

### Output:

Procedure created.

PL/SQL program to call procedure



1. **BEGIN**
2. insertuser(101,'Rahul');
3. dbms\_output.put\_line('record inserted successfully');
4. **END;**
5. /

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

| ID  | Name  |
|-----|-------|
| 101 | Rahul |

### PL/SQL Drop Procedure

#### Syntax for drop procedure

1. **DROP PROCEDURE** procedure\_name;

Example of drop procedure

1. **DROP PROCEDURE** pro1;

### EXAMPLE PROGRAMS FOR PROCEDURE AND FUNCTIONS

#### Example 1: Executing from a single named procedure

##### Step 1: Type the procedure coding

create or replace procedure divexp(a number, b number) as

c number;

begin

c := a/b;

dbms\_output.put\_line('The Result of C is : ' || c);

exception

when zero\_divide then

dbms\_output.put\_line('Divide by Zero Error !!!!');

end divexp;



/

**Output:**

Procedure created.

**Step 2: Setting the Server to output mode.**

SQL> set serveroutput on;

**Step 3: To Call the Procedure and to display the Value of C**

SQL> call divexp(144,3);

**Output:**

The Result of C is : 48

Call completed.

SQL> call divexp(28,0);

**Output:**

Divide by Zero Error !!!!

Call completed.

**Example 2: Calling a Procedure inside a procedure.**

**Step 1: Type the Coding**

SQL> create or replace

2 procedure sum\_ab (a int, b int, c out int) is

3 begin

4 c := a + b;

5 end;

6 /

**Output:**

Procedure created.





**Step 2:**

SQL> set serveroutput on;

**Step 3: Type the Calling Procedure**

SQL> declare

```
2 r int;
3 begin
4 sum_ab(23,29,r);
5 dbms_output.put_line('sum is: ' || r);
6 end;
7 /
```

**Output:**

SUM IS: 52

PL/SQL procedure successfully completed.

**Example 3: Using Table fields.**

SQL> select \* from stu;

| RNO | NAME  | DBMS | DS | CO | TOTAL | AVERAGE | RESULT |
|-----|-------|------|----|----|-------|---------|--------|
| 1   | shree | 90   | 90 | 90 | 270   | 90      | PASS   |
| 2   | priya | 80   | 80 | 80 | 240   | 80      | PASS   |
| 3   | anu   | 20   | 30 | 56 | 106   | 35      | FAIL   |
| 4   | suja  | 67   | 34 | 56 | 157   | 52      | FAIL   |

**Step 1: Type the coding in SQL prompt.**

create or replace procedure tabproc(no number)as

```
m1 number;
m2 number;
m3 number;
tot number;
```



```
aver number;
failexp exception;
begin
select dbms,ds,co into m1,m2,m3 from stu where rno=no;
if m1<50 or m2<50 or m3<50 then
raise failexp;
else
tot :=m1+m2+m3;
aver :=tot/3;
dbms_output.put_line('TOTAL : ' ||tot);
dbms_output.put_line('AVERAGE : ' ||aver);
dbms_output.put_line('The Student is Passed in all the subject');
end if;
exception
when failexp then
dbms_output.put_line('The Student is Failed');
when others then
dbms_output.put_line('LOOP is not executed');
end tabproc;
/
```

**Output:**

Procedure created.

**Step 2: Call the procedure by giving the input.**

```
SQL> call tabproc(4);
```

**Output:**

The Student is Failed

Call completed.

```
SQL> call tabproc(1);
```



**Output:**

TOTAL : 270

AVERAGE : 90

The Student is Passed in all the subject

Call completed.

**FUNCTION**

**Example 1: Simple Function**

**Step 1: Type the coding.**

create or replace function add\_two (a int,b int) return int is

begin

return (a + b);

end;

**Output:**

Function created

**Step 2: To run the coding call the above function by typing the following coding.**

begin

dbms\_output.put\_line('RESULT IS : ' || add\_two(12,34));

end;

**Output:**

RESULT IS : 46

PL/SQL procedure successfully completed.

**Example 2: Using Table.**

**Step 1: Type the coding**

create or replace function tabfunc(no number) return varchar2 as

m1 number;

m2 number;

m3 number;

tot number;



```
aver number;
failexp exception;
begin
select dbms,ds,co into m1,m2,m3 from stu where rno=no;
if m1<50 or m2<50 or m3<50 then
raise failexp;
else
tot :=m1+m2+m3;
aver :=tot/3;
dbms_output.put_line('TOTAL : ' ||tot);
dbms_output.put_line('AVERAGE : ' ||aver);
return('PASS');
end if;
exception
when failexp then
return ('FAIL');
when others then
dbms_output.put_line('LOOP is not executed');
end;
/
```

**Output:**

Function created.

**Step 2:**

SQL> set serveroutput on;

**Step 3: To pass the values for the above function.**

```
SQL> begin
 dbms_output.put_line('RESULT IS : ' || tabfunc(1));
end;
/
```



**Output:**

TOTAL : 270

AVERAGE : 90

RESULT IS : PASS

PL/SQL procedure successfully completed.

SQL> begin

```
 dbms_output.put_line('RESULT IS : ' || tabfunc(4));
```

```
 end;
```

```
 /
```

**Output:**

RESULT IS : FAIL

PL/SQL procedure successfully completed.

\*\*\*\*\*

**Review Questions**

1. What is the FUNCTION operation? What is it used for?
2. Why do we use database triggers? Give the syntax of a trigger.
3. Explain exception handling in PL/SQL.
4. How is a DECLARE statement used?
5. How do you compile PL/SQL code?
6. What is the difference between %TYPE and %ROWTYPE? Give an example.
7. List some schema objects that are created using PL/SQL.
8. Explain the difference between procedure and function.
9. What are the differences between triggers and constraints?
10. What is rollback? How is it different from rollback to statement?
11. What are some of the pre-defined exceptions in PL/SQL?
12. Can you use IF statement inside a SELECT statement? How?
13. Write a simple procedure to select some records from the database using some parameters.

\*\*\*\*\*



### OBJECTIVE TYPE QUESTIONS

1. Which of the following is true about data types in PL/SQL?
  - a) Large Object or LOB data types are pointers to large objects that are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms.
  - b) The composite data types have data items that have internal components that can be accessed individually. For example, collections and records.
  - c) References are pointers to other data items.
  - d) All of the above
2. Which of the following is not true about the PL/SQL data structure VARRAY?
  - a) In oracle environment, the starting index for VARRAYs is always 1.
  - b) You can initialize the VARRAY elements using the constructor method of the VARRAY type, which has the same name as the VARRAY.
  - c) VARRAYs are one-dimensional arrays.
  - d) None of the above.

#### 3. DECLARE

```
-- Global variables
num number := 95;
```

#### BEGIN

```
dbms_output.put_line('num: ' || num1);
```

#### DECLARE

```
-- Local variables
num number := 195;
```

#### BEGIN

```
dbms_output.put_line('num: ' || num1);
```

#### END;

#### END

**What will happen when the code is executed?**

- a) It won't execute, it has syntax error
- b) It will print num: 95 num: 195
- c) It will print num: 95 num: 95
- d) It will print num: 195 num: 195

#### 4. What is wrong in the following code snippet?

#### DECLARE

```
x number := 1;
```

#### BEGIN

#### LOOP

```
dbms_output.put_line(x);
```



```
x := x + 1;
IF x > 10 THEN
 exit;
END IF;
dbms_output.put_line('After Exit x is: ' || x);
END;
```

- a) There is nothing wrong.
- b) The IF statement is not required.
- c) There should be an END LOOP statement.
- d) The exit statement should be in capital letters.

5 - Which of the following is not true about PL/SQL cursors?

- a) A cursor is a view on a table.
- b) A cursor holds the rows (one or more) returned by a SQL statement.
- c) The set of rows the cursor holds is referred to as the active set.
- d) None of the above.

6. The pre-defined exception **NO\_DATA\_FOUND** is raised when

- a) A null object is automatically assigned a value.
- b) A SELECT INTO statement returns no rows.
- c) PL/SQL has an internal problem.
- d) PL/SQL ran out of memory or memory was corrupted.

7 - Observe the syntax given below –

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

Which of the following holds true for the WHEN clause?



- a) This provides a condition for rows for which the trigger would fire and this clause is valid only for row level triggers.
- b) This provides a condition for rows for which the trigger would fire and this clause is valid only for table level triggers.
- c) This provides a condition for rows for which the trigger would fire and this clause is valid only for view based triggers.
- d) All of the above.

**8. All objects placed in a package specification are called**

- a) Public objects.
- b) Private objects.
- c) None of the above.
- d) Both of the above.

**9. Which of the following code is the correct syntax for creating an index-by table named salary that will store integer values along with names and the name field will be the key?**

- a) TYPE salary IS TABLE OF NUMBER INDEX BY VARCHAR2(20);
- b) CREATE TABLE salary OF NUMBER INDEX BY VARCHAR2(20);
- c) TYPE salary IS INDEXED TABLE OF NUMBER INDEX BY VARCHAR2(20);
- d) None of the above.

**10. Savepoints are set to**

- a) Help in splitting a long transaction into smaller units.
- b) Help in rolling back to some checkpoint, within a long transaction.
- c) To execute a COMMIT automatically.
- d) Answer a. and b.

**11. Which of the following is not true about PL/SQL cursors?**

- a) A cursor is a view on a table.
- b) A cursor holds the rows (one or more) returned by a SQL statement.
- c) The set of rows the cursor holds is referred to as the active set.
- d) None of the above.

**12. The pre-defined exception NO\_DATA\_FOUND is raised when**

- a) A null object is automatically assigned a value.
- b) A SELECT INTO statement returns no rows.
- c) PL/SQL has an internal problem.
- d) PL/SQL ran out of memory or memory was corrupted.

**13. Which Exception is also known as Oracle named exception handler?**

- a) Predefined Exception
- b) Internal Exception
- c) User defined Exception





- d) None of the above
- 14. Which Package lets you use database triggers to alert an application when specific database values change?**
- a) DBMS\_OUTPUT
  - b) DBMS\_ALERT
  - c) DBMS\_PIPE
  - d) All mentioned above
- 15. Which statement lets you create standalone functions that are stored in an Oracle database?**
- a) SQL CREATE PROCEDURE
  - b) SQL CREATE FUNCTION
  - c) Both A & B
  - d) None of the above
- 16. Which parameter acts like a constant inside the subprogram?**
- a) IN
  - b) OUT
  - c) Both A & B
  - d. None of the above
- 17. Which of the following is used to define code that is executed / fired when certain actions or event occur?**
- a) Replace
  - b) Keyword
  - c) Trigger
  - d) Cursor
- 18. Which Operator Returns TRUE if a subquery returns at least one row?**
- a) EXISTS
  - b) IN
  - c) IS NULL
  - d) LIKE
- 19. How many attributes does every explicit cursor and cursor variable have?**
- a) 3
  - b) 2
  - c) 4
  - d) 5
- 20. Which datatypes can be used with a RECORD Type?**



திருவள்ளூர் பல்கலைக்கழகம்  
**THIRUVALLUVAR UNIVERSITY**

(State University Accredited with "B" Grade by NAAC)  
Serkkadu, Vellore - 632 115, Tamil Nadu, India.

E-NOTES / CS & BCA

- a) NUMBER, VARCHAR2
- b) %TYPE, OR %ROWTYPE
- c) REF, CURSOR
- d) BOTH A & B

**KEYS**

1-d, 2-d, 3-b, 4-c, 5-a, 6-b, 7-a, 8-a, 9-a, 10-d, 11-a, 12-b, 13-a, 14-b,  
15-b, 16-a, 17-c, 18-a, 19-c, 20-d.

\*\*\*\*\*



## CHAPTER 9: OBJECT ORIENTED TECHNOLOGY

### 9.1. INTRODUCTION

**Object-oriented technology** (OOT) is a software design model in which **objects** contain both data and the instructions that work on the data. It is increasingly deployed in distributed computing.



An Object Oriented Database (OODB) is a system combining characteristics of a database with the manipulation of objects typically available in object oriented languages. In object oriented database, information is represented in the form of objects.

Object oriented databases are exactly same as object oriented programming languages. If we can combine the features of relational model (transaction, concurrency, recovery) to object oriented databases, the resultant model is called as object oriented database model.



## 9.2. OBJECT ORIENTED FEATURES

- **Complex Objects:** these are built from simpler ones by applying constructors to them. Constructors must be orthogonal to the objects, meaning they can be applied to each object. The use of complex objects improves the capability of representing of the real world.
- **Object Identity:** OO systems are identity-based, meaning that each representation of information has its own identifiers. Please note that identity-based models are common in OO programming languages but rather new in database technology, since in most relational databases relations are valued based.
- **Encapsulation:** An object contains both programs and data and offers to the world an interface and an implementation part. The interface part is the specification of the set of operations that can be performed on the object; the implementation part describes the implementation of each operation. In most OODB, even data specification is part of the interface.
- **Types and Classes:** a type summarizes the common features of a set of objects; a class has an extension which contains the set of objects that instantiates the classes and has a set of operations with which the user can manipulate the objects.
- **Inheritance:** the ability of a subclass to receive all data and operations coming from its super classes. It helps code reusability and is also a better-structured and more concise description of the real world and the shared specifications of applications.
- **Overriding, Overloading and Late Binding:** when a single identifier is bound to different operation code in different types, one says that the code is overridden and the operation is overloaded. To provide this functionality, code is not bound to operation identifiers at compile time but at run time, thus performing the so called late binding.
- **Computational completeness:** the ability of expressing and computing functions using any combination of the available manipulating operations over the data is



natural for any programming language, but rather new for database languages. In this respect SQL is not complete.

- **Extensibility:** predefined and user-defined types must have the same status, meaning they must be supported by the system as completely equivalent.

### 9.3. COMPONENTS

1. Messages
2. Methods
3. Variables

#### 9.3.1. MESSAGES

A message provides an interface or acts as a communication medium between an object and the outside world. A message can be of two types:

- a. **Read-only message:** If the invoked method does not change the value of a variable, then the invoking message is said to be a read-only message.
- b. **Update message:** If the invoked method changes the value of a variable, then the invoking message is said to be an update message.

#### 9.3.2. METHODS

When a message is passed then the body of code that is executed is known as a method. Every time when a method is executed, it returns a value as output. A method can be of two types:

- a. **Read-only method:** When the value of a variable is not affected by a method, then it is known as read-only method.
- b. **Update-method:** When the value of a variable changes by a method, then it is known as an update method.

#### 9.3.3. VARIABLES

It stores the data of an object. The data stored in the variables makes the object distinguishable from one another.



#### 9.4. OBJECT CLASSES

An object which is a real world entity is an instance of a class. Hence first we need to define a class and then the objects are made which differ in the values they store but share the same class definition.

The objects in turn corresponds to various messages and variables stored in it.

##### **Example –**

```
class CLERK
{ //variables
 char name;
 string address;
 int id;
 int salary;
 //messages
 char get_name();
 string get_address();
 int annual_salary();
};
```

In above example we can see, CLERK is a class that holds the object variables and messages.

An OODBMS also supports inheritance in an extensive manner as in a database there may be many classes with similar methods, variables and messages. Thus, the concept of class hierarchy is maintained to depict the similarities among various classes.

The concept of encapsulation that is the data or information hiding is also supported by object oriented data model. And this data model also provides the facility of abstract data



types apart from the built-in data types like char, int, float. ADT's are the user defined data types that hold the values within it and can also have methods attached to it.

Thus, OODBMS provides numerous facilities to its users, both built-in and user defined. It incorporates the properties of an object oriented data model with a database management system, and supports the concept of programming paradigms like classes and objects along with the support for other concepts like encapsulation, inheritance and the user defined ADT's (abstract data types).

\*\*\*\*\*

### REVIEW QUESTIONS

1. What are the various kinds of interactions catered by DBMS?
2. Segregate database technology's development.
3. Enlist the various relationships of database.

\*\*\*\*\*

### OBJECTIVE TYPE QUESTIONS

**1. ODL supports which of the following types of association relationships?**

- a) Unary
- b) Unary and Binary
- c) Unary and Binary and Ternary
- d) Unary and Binary and Ternary and higher

**2. An extent is which of the following?**

- a) A keyword that indicates that the subclass inherits from a superclass
- b) A keyword that indicates that the superclass inherits from a subclass
- c) The set of all instances of a class within a database
- d) Only one instance of a class within a database

**3. Identify the class name for the following code: ABC123 course();**

- a) ABC123
- b) course
- c) course()
- d) All of the above

**4. Using ODL, you can define which of the following?**



- a) Attribute
- b) Structure
- c) Operation
- d) All of the above

**5. The keyword "inverse" is used in which of the following?**

- a) Class
- b) Attribute
- c) Relationship
- d) All of the above

**KEYS**

1-b, 2-c, 3-a, 4-d, 5-c

\*\*\*\*\*





## Appendix A: ODBC CONNECTIVITY (VB with SQL \* PLUS)

### Step 1 : Create a table in SQL\*Plus

SQL> desc stu

| Name    | Null? | Type         |
|---------|-------|--------------|
| RNO     |       | NUMBER(3)    |
| NAME    |       | VARCHAR2(20) |
| DBMS    |       | NUMBER(3)    |
| DS      |       | NUMBER(3)    |
| CO      |       | NUMBER(3)    |
| TOTAL   |       | NUMBER(4)    |
| AVERAGE |       | NUMBER(4)    |
| RESULT  |       | VARCHAR2(8)  |

Step 2 : Open Control Panel -> Administrative tools -> Data Sources (ODBC) -> Add -> Microsoft ODBC for oracle -> type the following

Microsoft ODBC for Oracle Setup

Data Source Name: priya OK

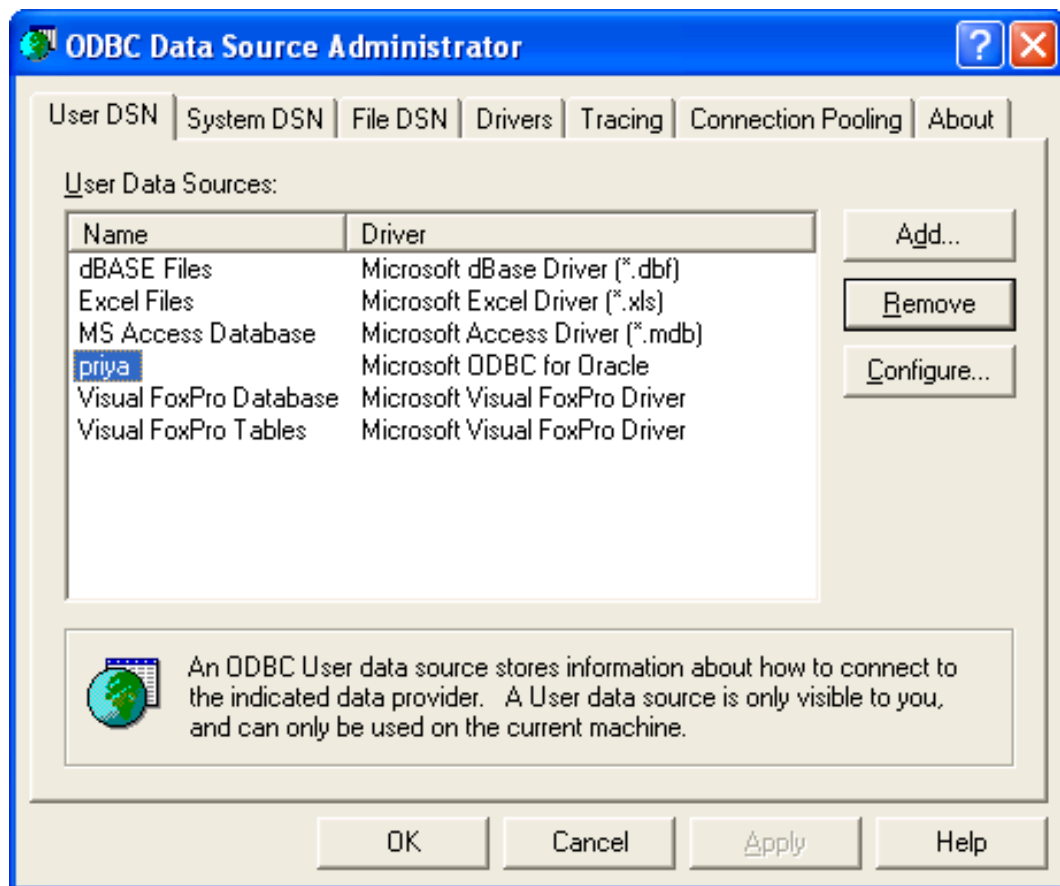
Description: Student Information System Cancel

User Name: scott Help

Server: Options >>

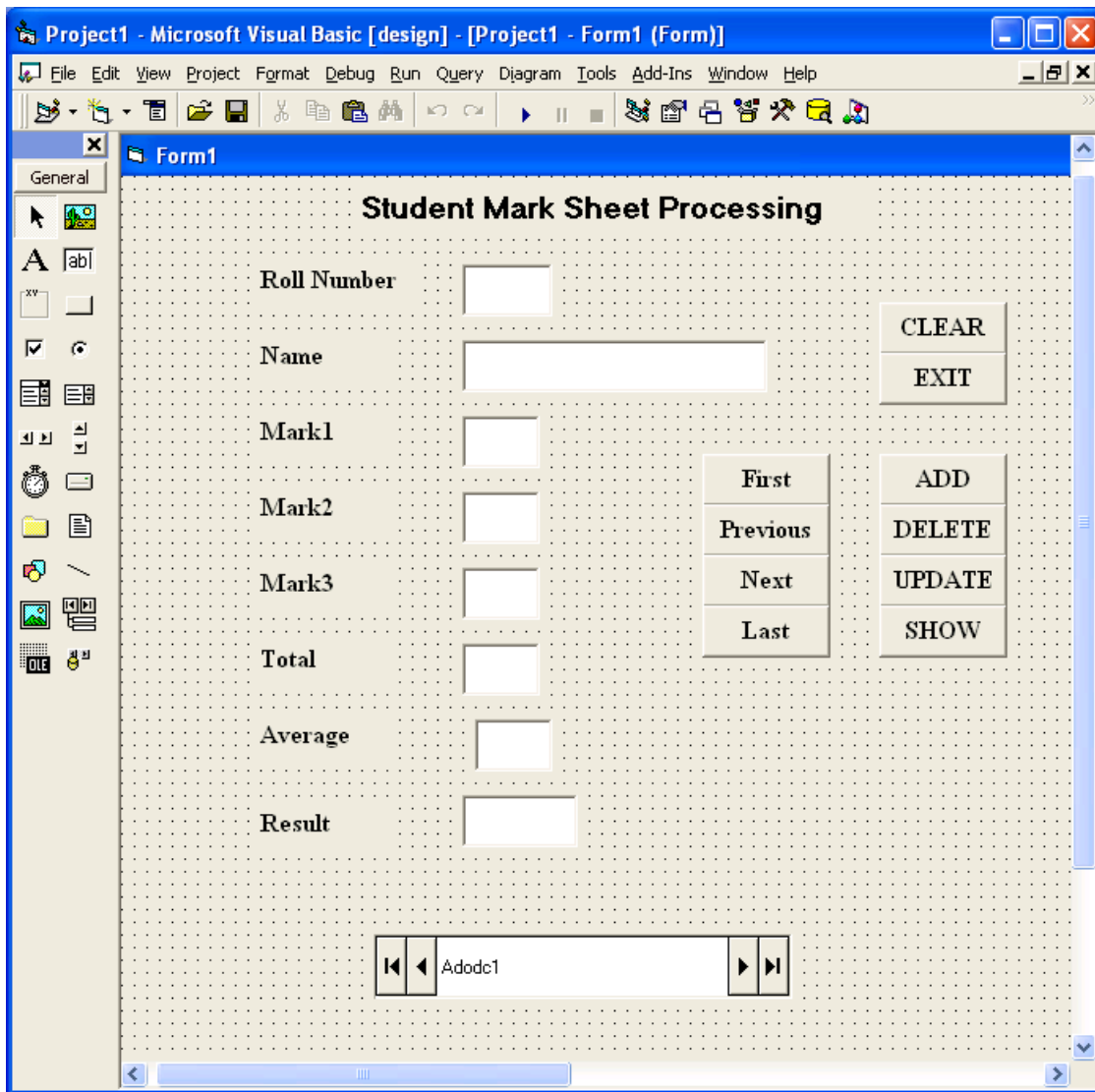
Then click OK

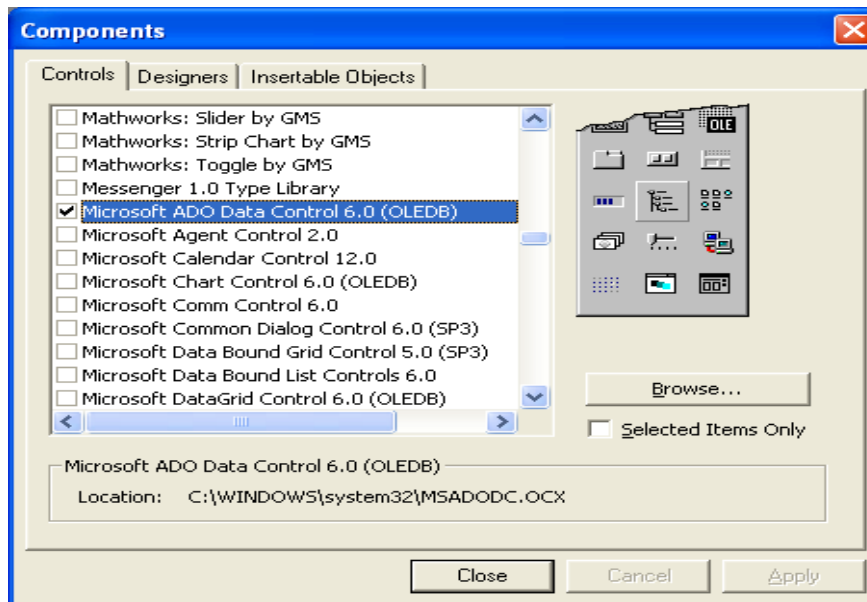
The following Screen will appear



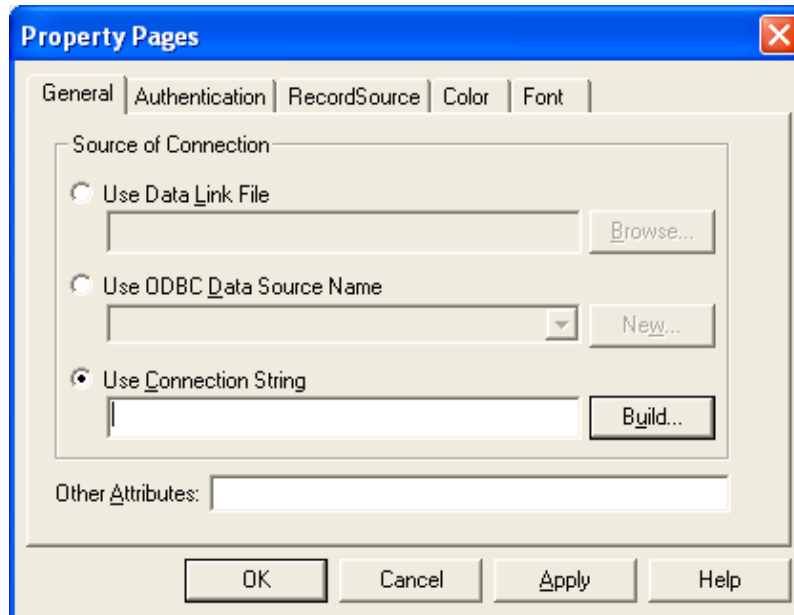
Then click OK

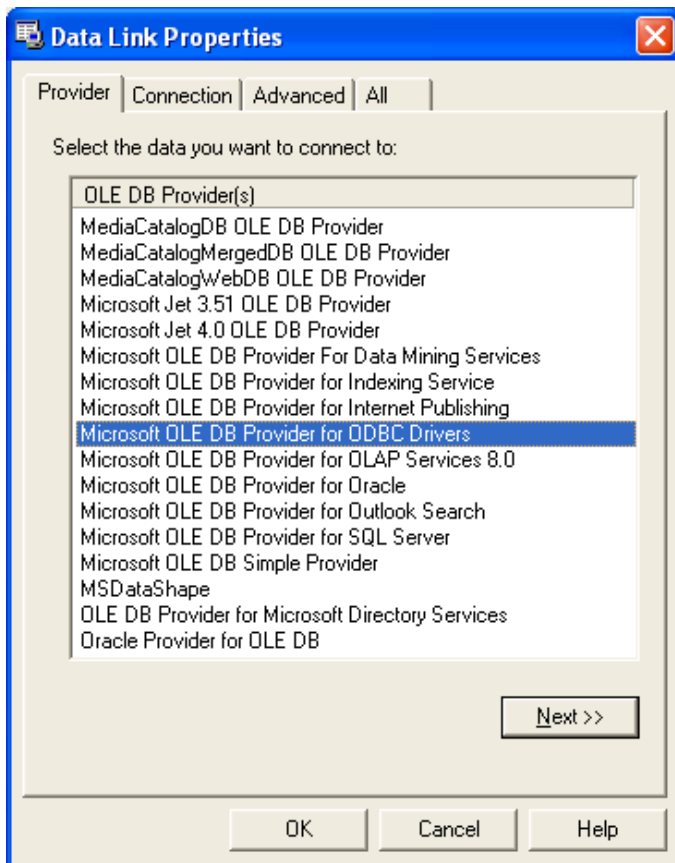
Step 3 : Now Open VB Form Screen. And Design your form as follows





Step 4 : Go to the Properties of Adodc1





Click Next



**Data Link Properties**

Provider | **Connection** | Advanced | All

Specify the following to connect to ODBC data:

1. Specify the source of data:

- Use data source name
- dBASE Files
- Excel Files
- GlobalCar
- MS Access Database
- priva**
- Visual FoxPro Database
- Visual FoxPro Tables

2. Enter the user name and password:

User name: \_\_\_\_\_

Password: \_\_\_\_\_

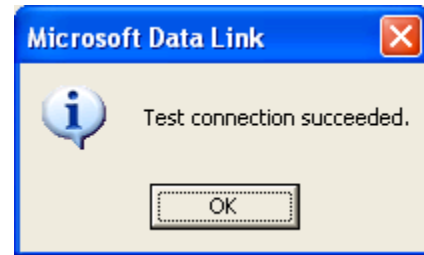
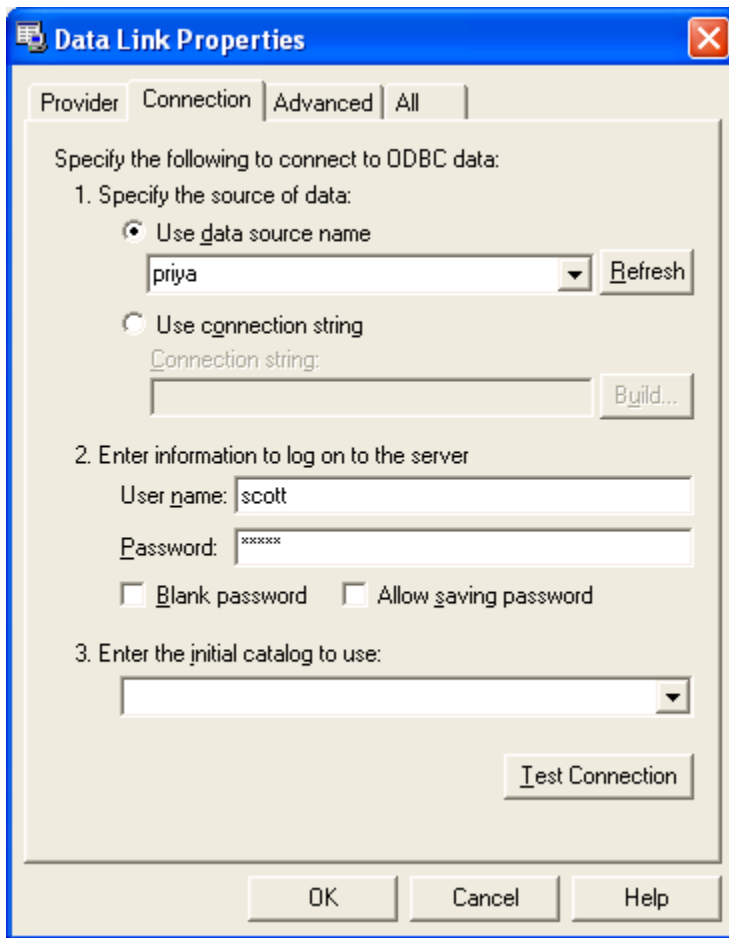
Blank password     Allow saving password

3. Enter the initial catalog to use:

\_\_\_\_\_

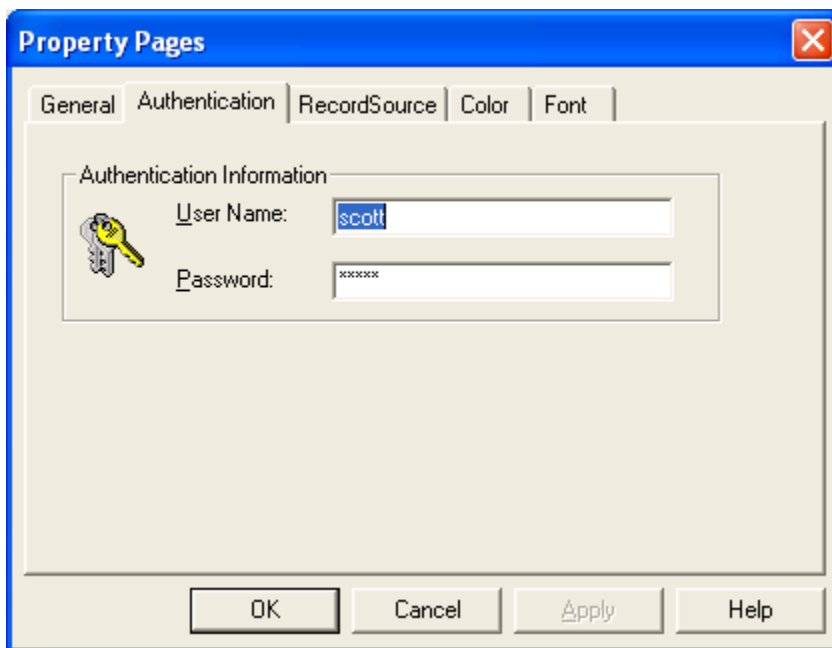
Refresh    Build...    Test Connection

OK    Cancel    Help

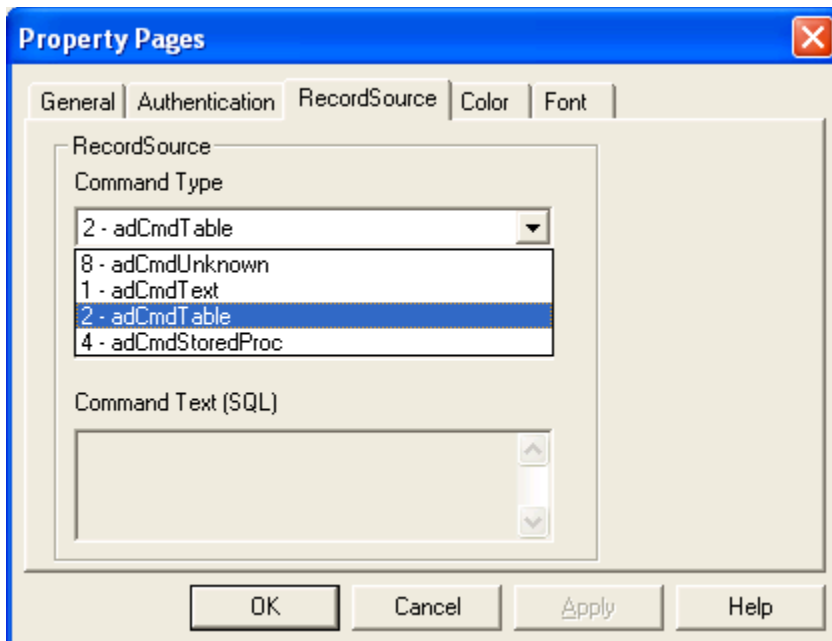


Click OK

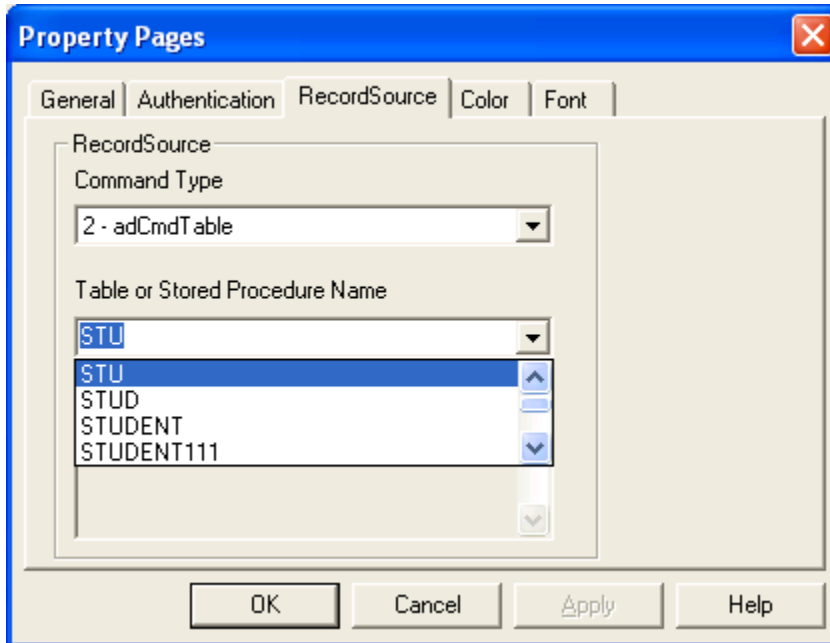
Go to Authentication and type as follows



Go to RecordSource and do the following

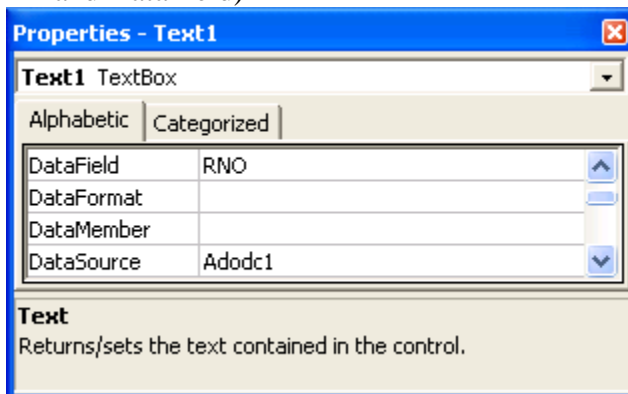






Then Finally Click OK

Step 5 : Now Change the properties of all the Text button to the following (DataSource and DataField)



Step 6 : After Changing all the properties of Text boxes type the Corresponding Code inn each buttons



Option Explicit

Dim db As New ADODB.Connection

Dim rs As New ADODB.Recordset

Dim query As String

---

Private Sub Form\_Load()

db.Open ("Provider=MSDASQL.1;User ID=scott;Password=tiger;Data Source=priya;Persist Security Info=False")

rs.Open "select \* from stu", db, adOpenDynamic

db.Properties.Refresh

End Sub

---

Private Sub Text5\_LostFocus()

Text6.Text = Val(Text3.Text) + Val(Text4.Text) + Val(Text5.Text)

Text7.Text = Val(Text6.Text) / 3

If Val(Text3.Text) >= 50 And Val(Text4.Text) >= 50 And Val(Text5.Text) >= 50 Then

Text8.Text = "PASS"

Else

Text8.Text = "FAIL"

End If

End Sub

---

Private Sub clear1()

Text1.Text = " "

Text2.Text = " "

Text3.Text = " "



Text4.Text = " "

Text5.Text = " "

Text6.Text = " "

Text7.Text = " "

Text8.Text = " "

End Sub

---

Private Sub display()

Text1.Text = rs(0)

Text2.Text = rs(1)

Text3.Text = rs(2)

Text4.Text = rs(3)

Text5.Text = rs(4)

Text6.Text = rs(5)

Text7.Text = rs(6)

Text8.Text = rs(7)

End Sub

---

Private Sub clear\_Click()

Call clear1

End Sub

---

Private Sub exit\_Click()

End

End Sub

---

Private Sub add\_Click()

query = "insert into stu values('" & (Text1.Text) & "','" & (Text2.Text) & "','" & (Text3.Text) & "','" & (Text4.Text) & "','" & (Text5.Text) & "','" & (Text6.Text) & "','" & (Text7.Text) & "','" & (Text8.Text) & "')"



```
db.Execute (query)
db.Execute ("commit")
MsgBox "SUCCESFULLY ADDED ", vbInformation + vbOKOnly, "Added"
Call clear1
db.Properties.Refresh
End Sub
```

---

```
Private Sub delete_Click()
query = "delete from stu where rno=" & (Text1.Text) & ""
db.Execute (query)
db.Execute ("commit")
MsgBox "Successfully Deleted the Record", vbInformation + vbOKOnly
Call clear1
db.Properties.Refresh
db.Close
Call Form_Load
End Sub
```

---

```
Private Sub update_Click()
query = "update stu set name=" & (Text2.Text) & ",m1=" & (Text3.Text) & ",m2=" &
(Text4.Text) & ",m3=" & (Text5.Text) & ",total=" & (Text6.Text) & ",average=" &
(Text7.Text) & ", result=" & (Text8.Text) & " where rollno=" & (Text1.Text) & ""
db.Execute (query)
db.Execute ("commit")
MsgBox "Successfully Modified the Record"
Call clear1
db.Properties.Refresh
```



db.Close

Call Form\_Load

End Sub

---

Private Sub show\_Click()

If Text1.Text = " " Then

MsgBox "Enter the Roll No to Display", vbInformation + vbOKOnly

Else

query = "select \* from stu where rno=" & (Text1.Text) & ""

Set rs = db.Execute(query)

Call display

End If

db.Properties.Refresh

db.Close

Call Form\_Load

End Sub

---

Private Sub first\_Click()

rs.MoveFirst

Call display

MsgBox "This is first record"

End Sub

---

Private Sub previous\_Click()

rs.MovePrevious

If rs.BOF Then



MsgBox "This is first record"

Else

Call display

End If

End Sub

---

Private Sub next\_Click()

rs.MoveNext

If rs.EOF = True Then

MsgBox "This is last record"

Else

Call display

End If

End Sub

---

Private Sub last\_Click()

rs.MoveLast

Call display

MsgBox "This is last record"

End Sub

Step 7 : Save & Executer your program.

\*\*\*\*\*

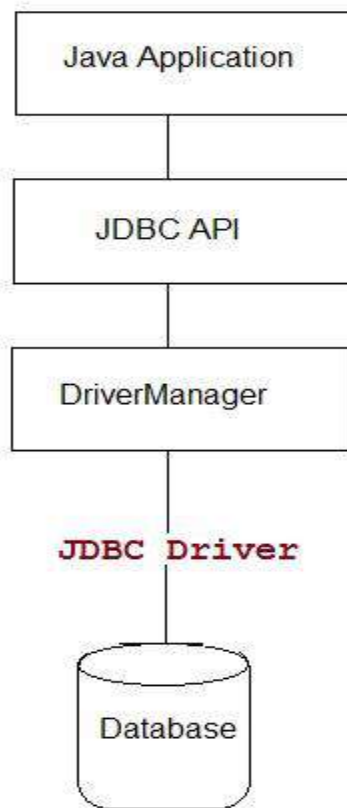


## Appendix B: JDBC CONNECTIVITY

JDBC Means Java Database Connectivity.

JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

Structure of JDBC as shown in the following figure



### Database

The database is defined as Organized Collection of related Information's.

### FMS

It means File Management System.



**Example:**

C, C++, COBOL, Fortran etc

To use File Management System (FMS)

Permanent Storage of Data (Data Consistency)

Drawback:

Security Less

Time Consume (Loop)

**DBMS**

Database Management System

**Example:**

DBase, FoxBASE, FoxPro, MS-Acess, Oracle, SQL Server etc

**Advantages:**

File Management System + Random Access Files

**Drawback:**

This system is Security Less because no implicit file formats.

**RDBMS**

Relational Database Management System

**Example:**

Oracle, Sybase, DB2, SQL-SERVER, MS-Access(partial)

**Data:** Collection of Information's (Raw & Facts). Anything can be Data or Data is meaningless

**Example:**

10    x    2000

**Example:**

| No | Name | Sal  |
|----|------|------|
| 10 | x    | 2000 |





### **File**

File means Collection of Records

### **Record**

File means Collection of Fields

### **Field**

File means Collection of Statements

### **Statement**

File means Collection of Words

### **Word**

File means Collection of Characters (Alphabets)

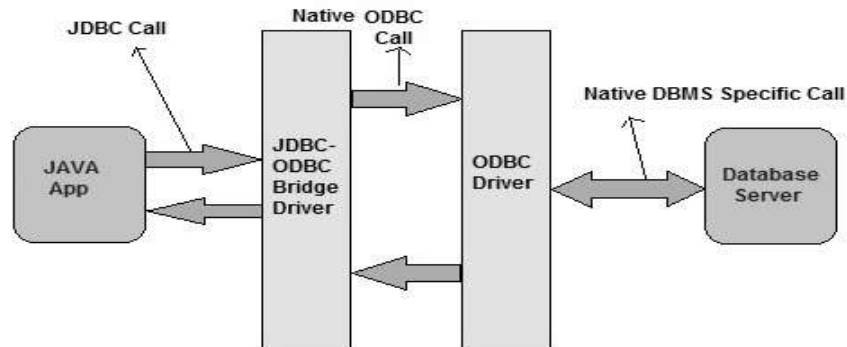
### **JDBC Driver**

JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- **Type-1 Driver** or **JDBC-ODBC bridge**
- **Type-2 Driver** or **Native API Partly Java Driver**
- **Type-3 Driver** or **Network Protocol Driver**
- **Type-4 Driver** or **Thin Driver**

### **DBC-ODBC Bridge**

**Type-1 Driver** act as a bridge between JDBC and other database connectivity mechanism (ODBC). This driver converts JDBC calls into ODBC calls and redirects the request to the ODBC driver.



### Advantage

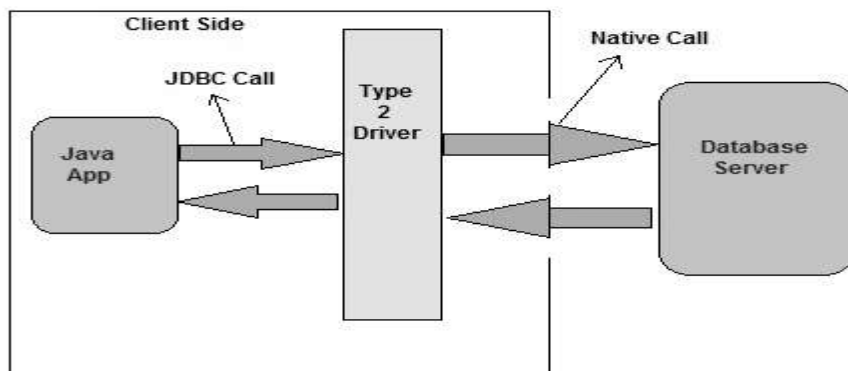
- Easy to use
- Allow easy connectivity to all database supported by the ODBC Driver.

### Disadvantage

- Slow execution time
- Dependent on ODBC Driver.
- Uses Java Native Interface (JNI) to make ODBC call.

### Native API Driver

This type of driver makes use of Java Native Interface (JNI) call on database specific native client API. These native clients API are usually written in C and C++.





### Advantage

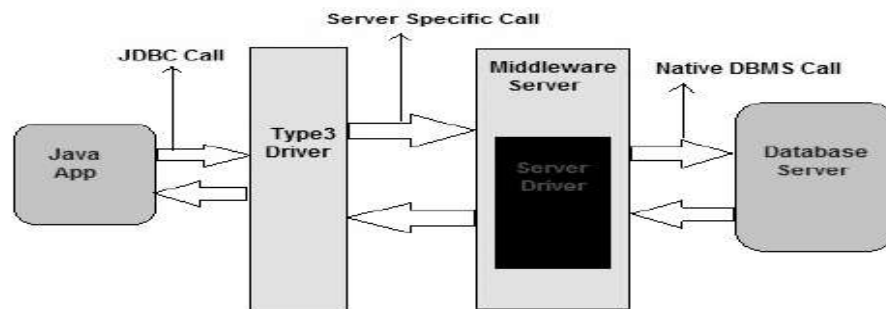
- faster as compared to **Type-1 Driver**
- Contains additional features.

### Disadvantage

- Requires native library
- Increased cost of Application

### Network Protocol Driver

This driver translates the JDBC calls into a database server independent and Middleware server-specific calls. Middleware server further translates JDBC calls into database specific calls.



### Advantage

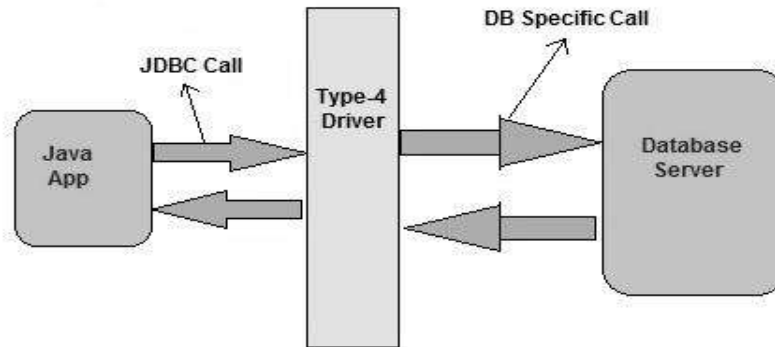
- Does not require any native library to be installed.
- Database Independency.
- Provide facility to switch over from one database to another database.

### Disadvantage

- Slow due to increase number of network call.

### Thin Driver

This is Driver called Pure Java Driver because. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.



### Advantage

- Does not require any native library.
- Does not require any Middleware server.
- Better Performance than other driver.

### Disadvantage

- Slow due to increase number of network call.

### Common JDBC Components

The JDBC API provides the following interfaces and classes

- **Driver Manager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain sub protocol under JDBC will be used to establish a database Connection.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use a DriverManager object, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.



- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterate to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

### **In java to connect with Database we have 5 Steps**

1. Load the Driver
2. Get the Connection
3. Create the Statement
4. Load the ResultSet
5. Close the Connection

java.sql.\*

This package Contains the Classes and Interfaces for Database concepts

#### **Note:**

All the Database concepts should raise SQLException  
(Or) to write the program in try catch block

### **1. Load the Driver**

JDK provides a Default driver to communicate with the Database which is available in JVM

"sun.jdbc.odbc.JdbcOdbcDriver"

java.lang.\*                      Default Package of Java

java.lang.Object                      Base class

Derived classes as follows

java.lang.String

java.lang.StringBuffer

.



java.lang.Class

This class contains a static method.

```
forName(String Driver);
```

```
*Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Through this step to load the driver

## 2. Get the Connection

It used to select the Database which going to use Connection Interface. It's in java.sql.\*

```
// Connection con = new Connection (); Error
```

```
java.sql.DriverManager
```

```
class
```

It contains a static Method

```
getConnection(String DSN) ret Connection
```

```
**Connection con = DriverManager.getConnection ("jdbc: odbc: DSN");
```

Through the above step we get the connection object

```
Connection
```

```
Interface
```

Methods:

```
createStatement() ret Statement
```

```
PreparedStaement() ret prepare Statement
```

```
close()
```

## DSN

It means Data Source Name.

## 3. Create the Statement

This statement is used to select the Table from the particular Database.



Statement                      Interface            in java.sql.\*;

\*\*\*Statement st = con.createStatement ();

3rd Step

Methods:

executeQuery (String DQL)    ret    ResultSet

executeQuery (String DML)    ret    int

executeQuery (String DDL)    ret    int

### SQL

It means Structured Query Language

Structure            It means Well Ordered

Query                It means retrieves the Information's

Language            it means Communication between user and Database

### SQL\*PLUS

It's an Editor

To work all the SQL/PL-SQL inside the SQL\*PLUS editor only

### RDBMS

**DDL:** It means Data Definition Language which contains following SQL commands.

CREATE, ALTER, DROP, TRUNCATE

**DML:** It means Data Manipulation Language which contains following SQL commands.

INSERT, UPDATE, DELETE & SELECT

**DCL:** It means Data Control Language which contains the following commands.

GRANT, REVOKE

**TCL:** It means Transaction Control Language which contains following commands.

COMMIT, ROLLBACK, SAVEPOINT



#### 4. Load the ResultSet

This step is used to fetch Records as per Query

Result Set Interface in java.sql.\*

ResultSet is a Collection

```
****ResultSet rs = st.executeQuery("select * from <Tname>");
```

#### Methods:

next ()            ret            boolean

getString (String value)

getString (int Field no, String value)

getString (String Field name)

getString (int Field no)

#### 5. Close the Connection

This step is used to save the transactions.

```
*****con.close();
```

#### Example Database

##### MS-Access

Tables in Database            No Limit

Records in Tables            No Limit

Fields in Table

in Oracle 7.x                    max 256

in Oracle >=8                  max 1000

f:\Sacet3\Tables.Mdb

Dept Table





### How to Create DSN?

Start->Settings->Control panel->ODBC Data Sources (32 Bit)

This step up to windows XP

Or

Start->Control Panel\System and Security\Administrative Tools

This step windows -7 to latest version

- A window will appear
- Click "Add" Button to create a new DSN
- Another window will Appear
- Select \*.mdb
- Click "Finish" Button
- Another window will appear
- Data Source Name Sact
- Description
- Click "Select" Button
- File Dialog will appear
- Select f:\Sacet3\Tables.mdb
- Click "Ok" Button
- Close the Control Panel

### Example 1: Program to show all records from Dept Table

```
import java.sql.*;

class sql
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con = DriverManager.getConnection("jdbc:odbc:Sact");
 Statement st = con.createStatement();
```



```
ResultSet rs = st.executeQuery("Select * from dept");
 while(rs.next())
 {
System.out.println(rs.getString("dno") + "\t" + rs.getString("dname") + "\t" +
rs.getString(3));
 }
 con.close();
} catch(Exception ex)
{
 System.out.println("sql : " + ex);
}
}
```

**Example 2: Program to show particular record from Dept Table**

```
import java.sql.*;
class sq2
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con = DriverManager.getConnection("jdbc:odbc:Sact");
 Statement st = con.createStatement();
 ResultSet rs = st.executeQuery("Select * from dept where dno='10'");
```



```
//ResultSet rs = st.executeQuery("Select * from dept where dno='10' or dno='20' or
dno='30'");
```

```
//ResultSet rs = st.executeQuery("Select * from dept where dno in('10','20','30')");
```

```
 while(rs.next())
 {
System.out.println(rs.getString("dno") + "\t" + rs.getString("dname") + "\t" +
rs.getString(3));
 }
 con.close();
 }catch(Exception ex)
 {
 System.out.println("sq2 : " + ex);
 }
}
}
```

### Example 3: Program to show particular record from Dept Table

```
import java.sql.*;
```

```
import java.io.*;
```

```
class sq3
```

```
{
```

```
 public static void main(String args[])
```

```
 {
```

```
 try
```



```
{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Connection con = DriverManager.getConnection("jdbc:odbc:Sact");

Statement st = con.createStatement();

 DataInputStream din = new DataInputStream(System.in);

 System.out.print("Enter Dno. : ");

 String no = din.readLine();

//ResultSet rs = st.executeQuery("Select * from dept where dno='no'");
ResultSet rs = st.executeQuery("Select * from dept where dno='"+no+"'");

 int i = 0;

 while(rs.next())
 {

System.out.println(rs.getString("dno") + "\t" + rs.getString("dname") + "\t" +
rs.getString(3));

 i = 1;

 }

 if(i == 0)
 {

 System.out.println("Dno Not Found");

 }

 con.close();

}
```



```
 }catch(Exception ex)
 {
 System.out.println("sq3 : " + ex);
 }
 }
}
```

**Example 4: Program to insert a record into Dept Table**

```
import java.sql.*;
import java.io.*;
class sq4
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con = DriverManager.getConnection("jdbc:odbc:Sact");
 Statement st = con.createStatement();
 //st.executeUpdate("insert into dept values('60','BBB','CCC')");
 DataInputStream din = new DataInputStream(System.in);

 System.out.print("Enter Dno. : ");
 String no = din.readLine();
 System.out.print("Enter Dna. : ");
 String na = din.readLine();
```



```
System.out.print("Enter Loc. : ");

String lc = din.readLine();

st.executeUpdate("insert into dept values('"+no+"','"+na+"','"+lc+"')");

System.out.println("1 Row Created");

con.close();
} catch(Exception ex)
{
 System.out.println("sq4 : " + ex);
}
}
```

### Example 5: Program to insert a Record into Dept Table using Prepared Statement

PreparedStatement is a Interface

#### Methods:

SetString (int Field no, String value)

executeUpdate ( )                      Insert, Update, Delete

```
import java.sql.*;
import java.io.*;
class sq5
{
 public static void main(String args[])
 {
 try
 {
```



```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Connection con = DriverManager.getConnection("jdbc:odbc:Sact");

PreparedStatement ps = con.prepareStatement("insert into dept values(?,?,?)");

 DataInputStream din = new DataInputStream(System.in);

 System.out.print("Enter Dno. : ");

 String no = din.readLine();

 System.out.print("Enter Dna. : ");

 String na = din.readLine();

 System.out.print("Enter Loc. : ");

 String lc = din.readLine();

ps.setString(1,no);

ps.setString(2,na);

ps.setString(3,lc);

ps.executeUpdate();

System.out.println("1 Row Created");

 con.close();

 }catch(Exception ex)

 {

 System.out.println("sq5 : " + ex);

 }

}

}
```

**Example 6: Program to Update a Record into Dept Table**

```
import java.sql.*;

import java.io.*;
```



```
class sq6
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con = DriverManager.getConnection("jdbc:odbc:Sact");
 Statement st = con.createStatement();

 DataInputStream din = new DataInputStream(System.in);

 System.out.print("Enter Dno. : ");
 String no = din.readLine();

 ResultSet rs = st.executeQuery("select * from dept where dno='"+no+"'");

 int i = 0;
 while(rs.next())
 {
 i = 1;
 break;
 }
 if(i == 1)
 {
 System.out.print("Enter Dna. : ");
 String na = din.readLine();

 System.out.print("Enter Loc. : ");
 String lc = din.readLine();
 }
 }
 }
}
```





```
st.executeUpdate("update dept set dname='"+na+"',loc='"+lc+"' where dno='"+no+"'");
 System.out.println("Updated");
 }
 else
 {
 System.out.println("Invalid DNo");
 }
 con.close();
} catch(Exception ex)
{
 System.out.println("sq6 : " + ex);
}
}
}
```

**Example 7: Program to Delete a Record from Dept Table**

```
import java.sql.*;
import java.io.*;
class sq7
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con = DriverManager.getConnection("jdbc:odbc:Sact");
```



```
Statement st = con.createStatement();

 DataInputStream din = new DataInputStream(System.in);
 System.out.print("Enter Dno. : ");
 String no = din.readLine();

ResultSet rs = st.executeQuery("select * from dept where dno='"+no+"'");
 int i = 0;
 while(rs.next())
 {
 i = 1;
 break;
 }

 if(i == 1)
 {
st.executeUpdate("delete from dept where dno='"+no+"'");
 System.out.println("Deleted");
 }
 else
 {
 System.out.println("Invalid DNo");
 }
 con.close();
}catch(Exception ex)
{
 System.out.println("sq7 : " + ex);
}
```



```
 }
 }
}
```

**Example 8: Program to Create a Table Emp into the Database**

```
import java.sql.*;
import java.io.*;
class sq8
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con = DriverManager.getConnection("jdbc:odbc:Sact");
 Statement st = con.createStatement();
 st.executeUpdate("create table Emp(Eno Text(5),Ename Text(25),Job Text(20), Sal
 Text(10),Dno Text(5))");
 System.out.println("Table Created");
 con.close();
 }catch(Exception ex)
 {
 System.out.println("sq8 : " + ex);
 }
 }
}
```



**Example 9: Program to show all Records from Scott.Emp Table in the Oracle Database**

```
import java.sql.*;2
import java.io.*;
class sq9
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con = DriverManager.getConnection("jdbc:odbc:Sact1","scott","tiger");
 Statement st = con.createStatement();
 ResultSet rs = st.executeQuery("select * from emp");
 while(rs.next())
 {
 System.out.println(rs.getString(1) + "\t" + rs.getString(2) + "\t" +rs.getString(3) + "\t" +
 rs.getString(6));
 }
 con.close();
 }catch(Exception ex)
 {
 System.out.println("sq9 : " + ex);
 }
 }
}
```



}

### Example 10: Applet using JDBC

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.sql.*;

public class appjdbc extends Applet implements ItemListener
{
 Choice c1 = new Choice();
 TextField t1 = new TextField(20);
 TextField t2 = new TextField(20);
 Connection con;
 Statement st;
 ResultSet rs;
 public void init()
 {
 add(c1); add(t1); add(t2);

 try
 {
 db();

 rs = st.executeQuery("select * from dept");
 while(rs.next())
 {
 c1.add(rs.getString(1));
 }
 }
 }
}
```



```
 con.close();
 }catch(Exception ex)
 {
 System.out.println("appjdbc init() : " + ex);
 }
 c1.addItemListener(this);
}
public void itemStateChanged(ItemEvent e)
{
 try
 {
 db();
//rs = st.executeQuery("select * from dept where dno="+c1.getSelectedItem()+""");
rs = st.executeQuery("select * from dept where deptno=" + c1.getSelectedItem());
 while(rs.next())
 {
 t1.setText(rs.getString(2));
 t2.setText(rs.getString(3));
 break;
 }
 con.close();
 }catch(Exception ex)
 {
 System.out.println("appjdbc ischg() : " + ex);
 }
}
```



```
public void db()
{
 try
 {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
// con = DriverManager.getConnection("jdbc:odbc:Sact");
con = DriverManager.getConnection("jdbc:odbc:Sact1","scott","tiger");
 st = con.createStatement();
 }catch(Exception ex)
 {
 System.out.println("appjdbc db() : " + ex);
 }
 }
 }
}
// <applet code=appjdbc width=300 height=300></applet>
```

### Example 11: Frames using JDBC

```
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
class frjdbc extends Frame implements ActionListener
{
 Panel p1 = new Panel(new GridLayout(3,2)); // Labels & Text
 Panel p2 = new Panel(new GridLayout(1,4)); // buttons
 Panel p3 = new Panel(new GridLayout(2,1)); // p1,p2
 Panel p4 = new Panel(new BorderLayout()); // p3
```



```
Label l1 = new Label("Enter Dno. : ");
TextField t1 = new TextField(20);
Label l2 = new Label("Enter Dna. : ");
TextField t2 = new TextField(20);
Label l3 = new Label("Enter Loc. : ");
TextField t3 = new TextField(20);
Button b1 = new Button("Save");
Button b2 = new Button("Edit");
Button b3 = new Button("Delete");
Button b4 = new Button("Exit");
Connection con;
Statement st;
ResultSet rs;
int i;
public frjdbc()
{
 p1.add(l1); p1.add(t1);
 t1.addFocusListener(new FocusAdapter()
 {
 public void focusLost(FocusEvent e)
 {
 try
 {
 db();
 }
 }
 });
 rs = st.executeQuery("select * from dept where dno='"+t1.getText()+"'");
 i = 0;
```





```
while(rs.next())
{
 t2.setText(rs.getString(2));
 t3.setText(rs.getString(3));
 i = 1;
 break;
}
con.close();
 }catch(Exception ex)
 {
 System.out.println("frjdbc const() : " + ex);
 }
 }
});
p1.add(l2); p1.add(t2);
p1.add(l3); p1.add(t3);
p2.add(b1); p2.add(b2); p2.add(b3);
p2.add(b4);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
p3.add(p1); p3.add(p2);
p4.add(p3, BorderLayout.CENTER);

addWindowListener(new WindowAdapter()
```



```
{

 public void windowClosing(WindowEvent e)
 {

 System.exit(-4);

 }

});
add(p4);
setSize(300,300);
setVisible(true);
}

public static void main(String args[])
{

 new frjdbc();

}
public void actionPerformed(ActionEvent e)
{

 if(e.getSource() == b1)
 {

 if(i == 0)
 {

 try
 {

 db();

st.executeUpdate("insertinto dept values (" +t1.getText()+"," +t2.getText()+",
"+t3.getText()+") ");

con.close();
```



```
t1.setText("");
t2.setText("");
t3.setText("");

t1.requestFocus();
 }catch(Exception ex)
 {
System.out.println("frjdbc apfm b1() : " + ex);
 }
 }
else if(e.getSource() == b2)
{
 if(i == 1)
 {
 try
 {
 db();
st.executeUpdate("update dept set dname='"+t2.getText()+"', loc='"+t3.getText()+"' where
dno='"+t1.getText()+"");
con.close();
t1.setText("");
t2.setText("");
t3.setText("");
t1.requestFocus();
i = 0;
 }catch(Exception ex)
 {
```



```
System.out.println("frjdbc apfm b2() : " + ex);
 }
 }
}

else if(e.getSource() == b3)
{
 if(i == 1)
 {
 try
 {
 db();
st.executeUpdate("delete from dept where dno="+t1.getText()+"");
con.close();
t1.setText("");
t2.setText("");
t3.setText("");
t1.requestFocus();
i = 0;
 }catch(Exception ex)
 {
System.out.println("frjdbc apfm b3() : " + ex);
 }
 }
}
else if(e.getSource() == b4)
```



```
{
 System.exit(4);
}
}
public void db()
{
 try
 {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 con = DriverManager.getConnection("jdbc:odbc:Sact");
 st = con.createStatement();
 }catch(Exception ex)
 {
 System.out.println("frjdbc db() : " + ex);
 }
}
}
```

**Example 12: Swings using JDBC loading all Records of Dept Table (Database) into the Swings Table**

```
import javax.swing.*;
import java.awt.*;
import java.sql.*;
import java.awt.event.*;
class swjdbc extends JFrame
{
 JTable t1;
```



```
Connection con;
Statement st;
ResultSet rs;

int i = 0;
public swjdbc()
{
 Container con1 = getContentPane();
 try
 {
 db();
rs = st.executeQuery("select * from dept");
 while(rs.next())
 {
 i++;
 }
 con.close();
 }catch(Exception ex)
 {
 System.out.println(ex);
 }
 String d[][] = new String[i][3];
 String t[] = {"Dno","Dname","Loc"};
 try
 {
 db();
```



```
int j = 0;

rs = st.executeQuery("select * from dept");

 while(rs.next())
 {

 d[j][0] = rs.getString(1);
 d[j][1] = rs.getString(2);
 d[j][2] = rs.getString(3);
 j++;
 }
 con.close();
}

catch(Exception ex)
{
 System.out.println(ex);
}

t1 = new JTable(d,t);

JScrollPane jsp = newJScrollPane
(t1,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);

 con1.add(jsp);

 setSize(300,300);

 setVisible(true);

}

public void db()
```



திருவள்ளூர் பல்கலைக்கழகம்  
**THIRUVALLUVAR UNIVERSITY**

(State University Accredited with "B" Grade by NAAC)  
Serkkadu, Vellore - 632 115, Tamil Nadu, India.

E-NOTES / CS & BCA

```
{
 try
 {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 con = DriverManager.getConnection("jdbc:odbc:Oss");
 st = con.createStatement();
 }catch(Exception ex)
 {
 System.out.println(ex);
 }
}
public static void main(String args[])
{
 new swjdbc();
}
}
```

\*\*\*\*\*





## APPENDIX C: SAMPLE PROGRAMS / APPLICATIONS using SQL \* PLUS

### 1. TABLE CREATION AND SIMPLE QUERIES

#### A. DATA DEFINITION LANGUAGE

**Aim:**

To execute data definition language using create, alter, drop.

#### 1. CREATE TABLE

**Problem:** Create a table with the fields reg, name, dob, m1, m2, m3, total, average.

**Query:**

```
SQL>create table student(reg number,name varchar (10),dob number(8),m1
number(5),m2 number(5),total number(5),average number(5,2)) ;
```

**Output:**

Table created.

#### 2. ALTER TABLE

**Problem:** Alter the table to add the field sex.

**Query:** SQL> alter table student add (sex varchar(2)) ;

**Output:**

Table altered.

**Query:** SQL> alter table student modify (dob number (10));

**Output:**

Table altered.

**Query:** SQL> alter table student drop column dob;

**Output:**

Column dropped

#### 3. DROP TABLE

**Problem:** Drop the created table using the drop query.

**Query:** SQL>drop table student;

**Output:**

Table dropped.

**Result:**

Thus the data definition language is executed successfully.



## **B. DATA MANIPULATION LANGUAGE**

### **Aim:**

To execute data manipulation language using select, update, insert, delete.

### **1. INSERT**

**Problem:** Insert fields in the corresponding table

**Query:** SQL> insert into student values ('&reg', '&name', '&dob', '&m1', '&m2', '&total', '&average');

Enter value for reg: 007

Enter value for name: Indhu

Enter value for dob: 01-jan-1989

Enter value for m1: 99

Enter value for m2: 99

Enter value for total: 198

Enter value for average: 99.00

old 1: insert into student values('&reg','&name','&dob','&m1','&m2','&total','&average')

new 1: insert into student values('007','Indhu','01-jan-1989','99','99','198','99.00')

insert into student values('007','Indhu','01-jan-1989','99','99','198','99.00')

### **Output:**

1 row inserted.

### **2. UPDATE**

**Problem:** Update the field using update query

**Query:** SQL> update student set name='priya' where reg='007';

### **Output:**

1 row updated.

### **3. DELETE**

**Problem:** Delete the record using delete query

**Query:** SQL> delete from student where reg=007;

### **Output:**

1 row deleted.



#### 4. SELECT

**Problem:** Select the table using select query

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

**Output:**

| REG NAME | DEPT  | M1  | M2 | M3 | TOTAL | AVG | RESULT | CLASS | FEES | AGE  |    |
|----------|-------|-----|----|----|-------|-----|--------|-------|------|------|----|
| 1        | Indhu | MCA | 99 | 99 | 99    | 297 | 99     | p     | a    | paid | 20 |
| 2        | Priya | MCA | 98 | 98 | 98    | 294 | 98     | p     | a    | paid | 21 |
| 3        | Divya | MBA | 97 | 97 | 97    | 291 | 97     | p     | a    | nil  | 29 |

**Problem:** Display the students record whose avg is > 97 and age = 20

**Query:** SQL> select \* from student where avg > 97 and age like '20';

**Output:**

| REG NAME | DEPT  | M1  | M2 | M3 | TOTAL | AVG | RESULT | CLASS | FEES | AGE  |    |
|----------|-------|-----|----|----|-------|-----|--------|-------|------|------|----|
| 1        | Indhu | MCA | 99 | 99 | 99    | 297 | 99     | p     | a    | paid | 20 |

**Problem:** Display the students record whose name ends with a.

**Query:** SQL> select \* from student where name like '%a';

**Output:**

| REG NAME | DEPT  | M1  | M2 | M3 | TOTAL | AVG | RESULT | CLASS | FEES | AGE  |    |
|----------|-------|-----|----|----|-------|-----|--------|-------|------|------|----|
| 2        | Priya | MCA | 98 | 98 | 98    | 294 | 98     | p     | a    | paid | 21 |

**Problem:** Display the students record whose total lies between 200 and 296.

**Query:** SQL> select \* from student where total between 200 and 296;

**Output:**

| REG NAME | DEPT  | M1  | M2 | M3 | TOTAL | AVG | RESULT | CLASS | FEES | AGE  |    |
|----------|-------|-----|----|----|-------|-----|--------|-------|------|------|----|
| 2        | Priya | MCA | 98 | 98 | 98    | 294 | 98     | p     | a    | paid | 21 |
| 3        | Divya | MBA | 97 | 97 | 97    | 291 | 97     | p     | a    | nil  | 29 |

**Problem:** Display the students record whose avg is > 98.



**Query:** SQL> select \* from student where avg > 98;

**Output:**

| REG NAME | DEPT  | M1  | M2 | M3 | TOTAL | AVG | RESULT | CLASS | FEES | AGE  |    |
|----------|-------|-----|----|----|-------|-----|--------|-------|------|------|----|
| 1        | Indhu | MCA | 99 | 99 | 99    | 297 | 99     | p     | a    | paid | 20 |

**Result:**

Thus the data manipulation language is executed successfully.

### C. SIMPLE QUERIES

**Aim:** To execute Simple queries using SQL.

#### a) Relational Operator

**Problem:** Display the students record whose avg is > 90

**Query** SQL> select \* from stud01 where avg>90;

**Output**

| ROLLNO | NAME  | MARK1 | MARK2 | TOTAL | AVG |
|--------|-------|-------|-------|-------|-----|
| 2      | priya | 91    | 95    | 186   | 93  |
| 3      | bala  | 95    | 92    | 187   | 94  |

**Problem:** Display the students record whose avg is < 80.

**Query**

SQL> select \* from stud01 where avg<80;

**Output**

| ROLLNO | NAME | MARK1 | MARK2 | TOTAL | AVG |
|--------|------|-------|-------|-------|-----|
| 1      | uma  | 85    | 70    | 155   | 75  |

**Problem:** Display the students record whose mark1 >= 95.

**Query:** SQL> select \* from stud01 where mark1>=95;



**Output**

| ROLLNO | NAME | MARK1 | MARK2 | TOTAL | AVG |
|--------|------|-------|-------|-------|-----|
| 3      | bala | 95    | 92    | 187   | 94  |

**Problem:** Display the students record whose mark1 <= 75.

**Query:** SQL> select \* from stud01 where mark2<=75;

**Output**

| ROLLNO | NAME | MARK1 | MARK2 | TOTAL | AVG |
|--------|------|-------|-------|-------|-----|
| 1      | uma  | 85    | 70    | 155   | 75  |

**Problem:** Display the students record whose total =187.

**Query:** SQL> select \* from stud01 where total=187;

**Output**

| ROLLNO | NAME | MARK1 | MARK2 | TOTAL | AVG |
|--------|------|-------|-------|-------|-----|
| 3      | bala | 95    | 92    | 187   | 94  |

**Problem:** Display the student's record whose total not equal to 190.

**Query:** SQL> select \* from stud01 where total ^!=190;

**Output**

| ROLLNO | NAME  | MARK1 | MARK2 | TOTAL | AVG |
|--------|-------|-------|-------|-------|-----|
| 1      | uma   | 85    | 70    | 155   | 75  |
| 2      | priya | 91    | 95    | 186   | 93  |
| 3      | bala  | 95    | 92    | 187   | 94  |

**B) Special Operators**

**Problem:** Display the student's name whose rollno is either 1 or 2.

**Query:** SQL> select name from stud01 where rollno IN (1,2);



**Output**  
NAME

-----  
uma  
priya

**Problem:** Display the student's name whose rollno is not 1.

**Query**

SQL> select name from stud01 where rollno NOT IN (1);

**Output**  
NAME

-----  
priya  
bala

**Problem:** Display the students average whose name is "uma".

**Query :** SQL> select avg from stud01 where name LIKE ('uma');

**Output**  
AVG

-----  
75

**Problem:** Display the student's average whose name is not like "uma".

**Query:** SQL> select avg from stud01 where name NOT LIKE ('uma');

**Output**  
AVG

-----  
93  
94

**Problem:** Display the student's records whose total is ranges from 180 to 190.

**Query :** SQL> select \* from stud01 where total between 180 and 190;



**Output**

| ROLLNO | NAME  | MARK1 | MARK2 | TOTAL | AVG |
|--------|-------|-------|-------|-------|-----|
| 2      | priya | 91    | 95    | 186   | 93  |
| 3      | bala  | 95    | 92    | 187   | 94  |

**c) Logical operators**

**Problem:** Display the student's records whose average is greater than 80 and less than 90.

**Query:** SQL> select \* from stud01 where avg>80 and avg<95;

**Output:**

| ROLLNO | NAME  | MARK1 | MARK2 | TOTAL | AVG |
|--------|-------|-------|-------|-------|-----|
| 2      | priya | 91    | 95    | 186   | 93  |
| 3      | bala  | 95    | 92    | 187   | 94  |

**Problem:** Display the student's records whose average is greater than 74 or less than 94.

**Query:** SQL> select \* from stud01 where avg>74 or avg<94;

**Output:**

| ROLLNO | NAME  | MARK1 | MARK2 | TOTAL | AVG |
|--------|-------|-------|-------|-------|-----|
| 1      | uma   | 85    | 70    | 155   | 75  |
| 2      | priya | 91    | 95    | 186   | 93  |
| 3      | bala  | 95    | 92    | 187   | 94  |

**Problem:** Display the student's records whose average is not greater than 90.

**Query:**

SQL> select \* from stud01 where not avg>90;

**Output:**

| ROLLNO | NAME | MARK1 | MARK2 | TOTAL | AVG |
|--------|------|-------|-------|-------|-----|
| 1      | uma  | 85    | 70    | 155   | 75  |

**RESULT:**

Thus the simple queries using SELECT command has been successfully executed.

\*\*\*\*\*



## 2. QUERIES USING AGGREGATE FUNCTION AND SET OPERATIONS

### A. AGGREGATE FUNCTION

**Aim:**

To execute aggregate functions using SQL \* PLUS.

#### 1. DISTINCT

**Problem:** Display the department without repetition.

**Query :** SQL> select distinct (dept) from student;

**Output:**

```
distinct (dept)
 mca
```

#### 2. AVERAGE

**Problem:** Display the average of the total from the student table..

**Query:** SQL> Select avg (total) from student;

**Output:**

```
avg(total)
 260
```

#### 3. COUNT

**Problem:** Display the count of the name from the student table..

**Query:** SQL>Select count (name) from student;

**Output:**

```
count (name)
 5
```

#### 4. MINIMUM

**Problem:** Display the minimum value of the average from the student table..

**Query:** SQL>select min (average) from student;





**Output:**

min (average).  
183

**5. MAXIMUM**

**Problem:** Display the maximum value of the average from the student table..

**Query:** SQL>select max (average) from student;

**Output:**

max (average)  
183

**6. SUM**

**Problem:** Display the sum of the total from the student table..

**Query:** SQL>select sum (total) from student;

**Output:**

sum (total)  
1300

**Result:**

Thus the aggregate or group functions are executed successfully.

**B. SET OPERATORS USING SELECT COMMAND**

**Aim:**

To execute set operators using SQL \* PLUS.

The **First** table,

| ID | Name |
|----|------|
| 1  | abhi |
| 2  | adam |



The **Second** table,

| ID | Name    |
|----|---------|
| 2  | adam    |
| 3  | Chester |

### 1. UNION

**Problem:** Display the records by combining two tables using Union

**Query:**

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

**Output:**

| ID | Name    |
|----|---------|
| 1  | abhi    |
| 2  | adam    |
| 3  | Chester |

### 2. UNION ALL

**Problem:** Display all the records by combining two tables using Union

**Query:**

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```



**Output:**

| ID | NAME    |
|----|---------|
| 1  | abhi    |
| 2  | adam    |
| 2  | adam    |
| 3  | Chester |

**3. INTERSECT**

**Problem:** Display the records for common records in both the tables using Intersect

**Query:**

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

**Output:**

| ID | NAME |
|----|------|
| 2  | adam |

**4. MINUS**

**Problem:** Display the records which is present on first table is not on the second table.

**Query:**

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```



திருவள்ளூர் பல்கலைக்கழகம்  
**THIRUVALUVAR UNIVERSITY**

(State University Accredited with "B" Grade by NAAC)  
Serkkadu, Vellore - 632 115, Tamil Nadu, India.

E-NOTES / CS & BCA

**Output:**

| ID | NAME |
|----|------|
| 1  | abhi |

**Result:**

Thus set operators using select command completed successfully.

\*\*\*\*\*



### 3. TABLE CREATION WITH VARIOUS JOINS

**Aim:**

To execute multiple table queries using inner join,outerjoin

**INNERJOIN**

**Query:**

SQL> select \* from stud01;

**Output:**

| ROLLNO | NAME  | MARK1 | MARK2 | TOTAL | AVG |
|--------|-------|-------|-------|-------|-----|
| 1      | uma   | 85    | 70    | 155   | 75  |
| 2      | priya | 91    | 95    | 186   | 93  |
| 3      | bala  | 95    | 92    | 187   | 94  |

**Query:**

SQL> select \* from stud;

**Output:**

| REGNO | DOB       | FEES  | SNO |
|-------|-----------|-------|-----|
| 1001  | 21-JAN-89 | 50000 | 1   |
| 1002  | 02-MAY-88 | 35000 | 2   |
| 1003  | 15-DEC-88 | 50000 | 3   |
| 1004  | 12-FEB-89 | 50000 | 4   |
| 1005  | 19-MAY-88 | 25000 | 5   |
| 1006  | 15-JUN-88 | 30000 | 6   |

6 rows selected.

**Query:**

SQL> select rollno,name,avg from stud01,stud where stud01.rollno=stud.sno;

**Output:**

| ROLLNO | NAME  | AVG |
|--------|-------|-----|
| 1      | uma   | 75  |
| 2      | priya | 93  |
| 3      | bala  | 94  |



## OUTER JOINS

### Left Outer Join:

#### Query:

```
SQL> select name,mark1,mark2,avg,fees from stud01,stud where
stud01.rollno=stud.sno(+);
```

#### Output:

| NAME  | MARK1 | MARK2 | AVG | FEES  |
|-------|-------|-------|-----|-------|
| uma   | 85    | 70    | 75  | 50000 |
| priya | 91    | 95    | 93  | 35000 |
| bala  | 95    | 92    | 94  | 50000 |

### Right Outer Join:

#### Query:

```
SQL> select regno,dob,avg,fees from stud,stud01 where stud.sno(+)=stud01.rollno;
```

#### Output:

| REGNO | DOB       | AVG | FEES  |
|-------|-----------|-----|-------|
| 1001  | 21-JAN-89 | 75  | 50000 |
| 1002  | 02-MAY-88 | 93  | 35000 |
| 1003  | 15-DEC-88 | 94  | 50000 |

#### Result:

Thus the multiple table queries using inner join and outer join are executed successfully.

\*\*\*\*\*



#### 4. NESTED SUB QUERIES AND CORRELATED SUB QUERIES

**Aim:**

To execute sub queries using simple and correlated sub queries.

**I)SIMPLE QUERY:**

**Query:**

SQL> select \* from stud where fees>(select avg(fees) from stud);

**Output:**

| REGNO | DOB       | FEES  | SNO |
|-------|-----------|-------|-----|
| 1001  | 21-JAN-89 | 50000 | 1   |
| 1003  | 15-DEC-88 | 50000 | 3   |
| 1004  | 12-FEB-89 | 50000 | 4   |

**ii) CORRELATED SUB QUERY**

**Query:**

SQL> select sno,dob from stud where sno=(select rollno from stud01  
2 where stud.sno=stud01.rollno);

**Output:**

| SNO | DOB       |
|-----|-----------|
| 1   | 21-JAN-89 |
| 2   | 02-MAY-88 |
| 3   | 15-DEC-88 |

**Result:**

Thus the sub queries using simple and correlated sub queries are executed successfully.

\*\*\*\*\*



## 5. VIEW CREATION AND MANIPULATION

### Aim:

To execute the views and its manipulation.

### I) CREATION OF VIEW

#### Query:

```
SQL> create view vcomp as select no,name,quali from comp;
```

#### Output:

View created.

### II) INSERTING DATA INTO VIEW

#### Query:

```
SQL>insert into vcomp values(&no,'&name','&quali');
```

Enter value for no: 6

Enter value for name: saranya

Enter value for quali: mba

```
old 1: insert into vcomp values(&no,'&name','&quali')
```

```
new 1: insert into vcomp values(6,'saranya','mba')
```

#### Output:

1 row created.

### III) SELECTING THE VIEW

#### Query:

```
SQL> select * from vcomp;
```

#### Output:

| NO | NAME    | QUALI |
|----|---------|-------|
| 1  | prema   | mca   |
| 2  | indu    | mca   |
| 3  | saran   | mca   |
| 4  | muthu   | bca   |
| 5  | prabha  |       |
| 6  | saranya | mba   |

6 rows selected.

### IV) UPDATION IN THE VIEW

#### Query:

```
SQL>update vcomp set quali='bca'where no=5;
```





**Output:**

1 row updated.

**Query:**

```
SQL> SELECT * FROM VCOMP;
```

**Output:**

| NO | NAME    | QUALI |
|----|---------|-------|
| 1  | prema   | mca   |
| 2  | indu    | mca   |
| 3  | saran   | mca   |
| 4  | muthu   | bca   |
| 5  | prabha  | bca   |
| 6  | saranya | mba   |

6 rows selected.

**V) DROPPING THE VIEW**

**Query:**

```
SQL>drop vcomp;
```

**Output:**

View dropped

**Result:**

Thus the database objects using views and its manipulation is executed successfully.

\*\*\*\*\*



## 6. PL/SQL PROGRAM FOR CURSOR

### Aim:

To execute the cursor using PL/SQL Program.

### Step 1:

```
SQL> insert into stu values(&rno,'&name',&dbms,&ds,&co,&total,&average,'&result');
Enter value for rno: 3
Enter value for name: anu
Enter value for dbms: 20
Enter value for ds: 30
Enter value for co: 56
Enter value for total: null
Enter value for average: null
Enter value for result: null
old 1: insert into stu values(&rno,'&name',&dbms,&ds,&co,&total,&average,'&result')
new 1: insert into stu values(3,'anu',20,30,56,null,null,null)
```

1 row created.

```
SQL> /
```

```
Enter value for rno: 4
Enter value for name: suja
Enter value for dbms: 67
Enter value for ds: 34
Enter value for co: 56
Enter value for total: null
Enter value for average: null
Enter value for result: null
old 1: insert into stu values(&rno,'&name',&dbms,&ds,&co,&total,&average,'&result')
new 1: insert into stu values(4,'suja',67,34,56,null,null,null)
```

### Step 2:

```
SQL> select * from stu;
```

| RNO | NAME  | DBMS | DS | CO | TOTAL | AVERAGE | RESULT |
|-----|-------|------|----|----|-------|---------|--------|
| 1   | shree | 90   | 90 | 90 | 270   | 90      | pass   |
| 2   | priya | 80   | 80 | 80 | 240   | 80      | pass   |
| 3   | anu   | 20   | 30 | 56 |       |         |        |
| 4   | suja  | 67   | 34 | 56 |       |         |        |



**Step 3: Type the following coding.**

```
SQL> declare
2 cursor stucur is select * from stu;
3 st stucur%rowtype;
4 begin
5 open stucur;
6 loop
7 fetch stucur into st;
8 exit when stucur%notfound;
9 st.total := st.dbms+st.ds+st.co;
10 st.average := st.total/3;
11 if(st.dbms >=50 and st.ds >=50 and st.co >=50) then
12 st.result := 'PASS';
13 else
14 st.result := 'FAIL';
15 end if;
16 update stu set total=st.total, average=st.average, result=st.result where rno=st.rno;
17 end loop;
18 end;
19 /
```

**Output:**

PL/SQL procedure successfully completed.

**Step 4:**

SQL> select \* from stu;

| RNO | NAME  | DBMS | DS | CO | TOTAL | AVERAGE | RESULT |
|-----|-------|------|----|----|-------|---------|--------|
| 1   | shree | 90   | 90 | 90 | 270   | 90      | PASS   |
| 2   | priya | 80   | 80 | 80 | 240   | 80      | PASS   |
| 3   | anu   | 20   | 30 | 56 | 106   | 35      | FAIL   |
| 4   | suja  | 67   | 34 | 56 | 157   | 52      | FAIL   |

**Result:** Thus the cursor using PL/SQL Program is successfully executed.

\*\*\*\*\*



## 7. PL/SQL PROGRAM FOR PACKAGES

### Aim:

To execute the packages using PL/SQL Program.

### SQL>

```
CREATE OR REPLACE PACKAGE aa_pkg AUTHID DEFINER IS
 TYPE aa_type IS TABLE OF INTEGER INDEX BY VARCHAR2(15);
END;
/
CREATE OR REPLACE PROCEDURE print_aa (
 aa aa_pkg.aa_type
) AUTHID DEFINER IS
 i VARCHAR2(15);
BEGIN
 i := aa.FIRST;
 WHILE i IS NOT NULL LOOP
 DBMS_OUTPUT.PUT_LINE (aa(i) || ' ' || i);
 i := aa.NEXT(i);
 END LOOP;
END;
/
DECLARE
 aa_var aa_pkg.aa_type;
BEGIN
 aa_var('zero') := 0;
 aa_var('one') := 1;
 aa_var('two') := 2;
 print_aa(aa_var);
END;
/
```

### OUTPUT:

```
1 one
2 two
0 zero
```

**Result:** Thus the packages using PL/SQL Program is successfully executed.

\*\*\*\*\*



## 8. PL/SQL PROGRAM FOR TRIGGERS AND ITS TYPE

### Aim:

To execute the triggers using PL/SQL Program.

### Trigger using Update

#### Step 1: Create two tables ystudent & zstudent

##### Table 1:

```
SQL> create table zstudent (rno number(3), name varchar2(20), Dept varchar2(20));
```

Table created.

##### Table 2:

```
SQL> create table ystudent(rno number(3), name varchar2(20), Dept varchar2(20));
```

Table created.

#### Step 2: Insert values for both the tables

```
SQL> insert into ystudent values(&rno, '&name', '&dept');
```

```
SQL> insert into zstudent values(&rno, '&name', '&dept');
```

```
SQL> select * from ystudent;
```

| RNO | NAME | DEPT |
|-----|------|------|
|-----|------|------|

|   |       |     |
|---|-------|-----|
| 1 | priya | mca |
| 2 | shree | mba |
| 3 | kavi  | cse |

```
SQL>select * from zstudent;
```

| RNO | NAME | DEPT |
|-----|------|------|
|-----|------|------|

|    |      |     |
|----|------|-----|
| 11 | ravi | mca |
| 21 | bala | eee |
| 31 | ram  | it  |

#### Step 3: Type the coding for Trigger

```
SQL>create or replace trigger stuttrigger
```

```
before update of dept
```

```
on zstudent
```



```
for each row
begin
insert into ystudent values
(:old.rno,:old.name,:old.dept);
end;
/
```

**Output:**  
Trigger created.

**Step 4: Type the coding to update the second table**

```
SQL>update zstudent set dept='ECE' where rno=3;
```

1 row updated.

**Step 5: Show the Updation on both the tables.**

```
SQL> select * from ystudent;
```

| RNO | NAME  | DEPT |
|-----|-------|------|
| 1   | priya | mca  |
| 2   | shree | mba  |
| 3   | kavi  | cse  |
| 31  | ram   | it   |

```
SQL> select * from zstudent;
```

| RNO | NAME | DEPT |
|-----|------|------|
| 11  | ravi | mca  |
| 21  | bala | eee  |
| 31  | ram  | ECE  |

**Result:** Thus the triggers using PL/SQL Program is successfully executed.

\*\*\*\*\*



## 9. PL/SQL PROGRAM FOR PROCEDURES AND FUNCTIONS

### Aim:

To execute the procedure and functions using PL/SQL Program.

### a. PROCEDURES

#### 1: Executing from a single named procedure

##### Step 1: Type the procedure coding

```
create or replace procedure divexp(a number, b number) as
c number;
begin
c := a/b;
dbms_output.put_line('The Result of C is : ' || c);
exception
when zero_divide then
dbms_output.put_line('Divide by Zero Error !!!!');
end divexp;
/
```

##### Output:

Procedure created.

##### Step 2: Setting the Server to output mode.

```
SQL> set serveroutput on;
```

##### Step 3: To Call the Procedure and to display the Value of C

```
SQL> call divexp(144,3);
```

##### Output:

The Result of C is : 48  
Call completed.

```
SQL> call divexp(28,0);
```

##### Output:

Divide by Zero Error !!!!  
Call completed.

#### 2: Calling a Procedure inside a procedure.



### Step 1: Type the Coding

```
SQL> create or replace
2 procedure sum_ab (a int, b int, c out int) is
3 begin
4 c := a + b;
5 end;
6 /
```

### Output:

Procedure created.

### Step 2:

```
SQL> set serveroutput on;
```

### Step 3: Type the Calling Procedure

```
SQL> declare
2 r int;
3 begin
4 sum_ab(23,29,r);
5 dbms_output.put_line('sum is: ' || r);
6 end;
7 /
```

### Output:

SUM IS: 52

PL/SQL procedure successfully completed.

### 3: Using Table fields.

```
SQL> select * from stu;
```

| RNO | NAME  | DBMS | DS | CO | TOTAL | AVERAGE | RESULT |
|-----|-------|------|----|----|-------|---------|--------|
| 1   | shree | 90   | 90 | 90 | 270   | 90      | PASS   |
| 2   | priya | 80   | 80 | 80 | 240   | 80      | PASS   |
| 3   | anu   | 20   | 30 | 56 | 106   | 35      | FAIL   |
| 4   | suja  | 67   | 34 | 56 | 157   | 52      | FAIL   |

### Step 1: Type the coding in SQL prompt.

```
create or replace procedure tabproc(no number)as
m1 number;
m2 number;
```





```
m3 number;
tot number;
aver number;
failexp exception;
begin
select dbms,ds,co into m1,m2,m3 from stu where rno=no;
if m1<50 or m2<50 or m3<50 then
raise failexp;
else
tot :=m1+m2+m3;
aver :=tot/3;
dbms_output.put_line('TOTAL : '||tot);
dbms_output.put_line('AVERAGE : '||aver);
dbms_output.put_line('The Student is Passed in all the subject');
end if;
exception
when failexp then
dbms_output.put_line('The Student is Failed');
when others then
dbms_output.put_line('LOOP is not executed');
end tabproc;
/
```

**Output:**

Procedure created.

**Step 2: Call the procedure by giving the input.**

SQL> call tabproc(4);

**Output:**

The Student is Failed

Call completed.

SQL> call tabproc(1);

**Output:**

TOTAL : 270

AVERAGE : 90

The Student is Passed in all the subject

Call completed.

**b. FUNCTIONS**

**1: Simple Function**



**Step 1: Type the coding.**

```
create or replace function add_two (a int,b int) return int is
begin
return (a + b);
end;
```

**Output:**

Function created.

**Step 2: To run the coding call the above function by typing the following coding.**

```
begin
dbms_output.put_line('RESULT IS : ' || add_two(12,34));
end;
```

**Output:**

RESULT IS : 46  
PL/SQL procedure successfully completed.

**2: Using Table.**

**Step 1: Type the coding**

```
create or replace function tabfunc(no number) return varchar2 as
m1 number;
m2 number;
m3 number;
tot number;
aver number;
failexp exception;
begin
select dbms,ds,co into m1,m2,m3 from stu where rno=no;
if m1<50 or m2<50 or m3<50 then
raise failexp;
else
tot :=m1+m2+m3;
aver :=tot/3;
dbms_output.put_line('TOTAL : ' ||tot);
dbms_output.put_line('AVERAGE : ' ||aver);
return('PASS');
end if;
exception
when failexp then
return ('FAIL');
when others then
```



```
dbms_output.put_line('LOOP is not executed');
end;
```

/

**Output:**

Function created.

**Step 2:**

```
SQL> set serveroutput on;
```

**Step 3: To pass the values for the above function.**

```
SQL> begin
 dbms_output.put_line('RESULT IS : ' || tabfunc(1));
end;
/
```

**Output:**

TOTAL : 270  
AVERAGE : 90  
RESULT IS : PASS

PL/SQL procedure successfully completed.

```
SQL> begin
 dbms_output.put_line('RESULT IS : ' || tabfunc(4));
end;
/
```

**Output:**

RESULT IS : FAIL

PL/SQL procedure successfully completed.

**Result:** Thus the procedures and functions using PL/SQL Program is successfully executed.

\*\*\*\*\*