

Oracle for Developers

Lesson 1: Introduction to Programming

People matter, results count.



©2016 Capgemini. All rights reserved.
The information contained in this document is proprietary and
confidential. For Capgemini only.

Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
13-Nov-2008	1.0		Rajita Dhumal	Content Creation
14-Nov-2008	1.1		CLS team	Review
14-Jan-2010	1.2		Anu Mitra	Review
14-Jan-2010	1.2		Rajita Dhumal, CLS Team	Incorporating Review comments
25-Apr-2011	2.0		Anu Mitra	Integration refinements
18-Feb-2014	2.1		Sathiabama Ranganathan	Course Refinements



Copyright © Capgemini 2016. All Rights Reserved 2

Course Goals and Non Goals

- **Course Goals**

- To understand basic DBMS, and use SQL commands.

- **Course Non Goals**

- Nothing Specific.



Pre-requisites

- A proficiency level in familiarity with Windows.



Copyright © Capgemini 2016. All Rights Reserved. 4

Intended Audience

- Software Programmers
- Software Analysts



Day Wise Schedule

■ Day 1

- Lesson 1: Introduction to Database
- Lesson 2: Basics of SQL
- Lesson 3: Data Query Language

■ Day 2

- Lesson 4: Aggregate (Group) Functions
- Lesson 5: SQL (Single-row) functions
- Lesson 6: Joins and Sub-queries (Joins to be covered here)



Copyright © Capgemini 2016. All Rights Reserved



Day Wise Schedule

■ Day 3

- Lesson 6: Joins and Sub-queries (Sub-queries to be covered here)
- Lesson 7: Introduction to Data Modeling, ER Modeling and Normalization
- Lesson 8: Database Objects (Table, Index, Sequence, Synonym to be covered here)

■ Day 4

- Lesson 8: Database Objects (Views to be covered here)
- Lesson 9: Set Operators
- Lesson 10: Data Manipulation Language
- Lesson 11: Transaction Control Language
- Lesson 12: Data Control Language



Copyright © Capgemini 2016. All Rights Reserved. 7

Table of Contents

- Lesson 1: Getting Started with Database
 - 1.1: Introduction to Database
 - 1.2: Characteristics of DBMS
 - 1.3: Data models
 - 1.4: Relational DBMS
 - 1.5: Database Administrator
- Lesson 2: Basics of SQL
 - 2.1. The SQL Language
 - 2.2. Rules for SQL Statements
 - 2.3. Standard SQL Statement Groups



Copyright © Capgemini 2016. All Rights Reserved. 8

Table of Contents

- Lesson 2: Basics of SQL (contd)
 - 2.4: Logging to Oracle Server
- Lesson 3: Data Query Language
 - 3.1: The SELECT statement
 - 3.2: The WHERE clause
 - 3.3: Comparison, Mathematical, and Logical operators
 - 3.4: The DISTINCT clause
 - 3.5: The ORDER BY clause
 - 3.6: Tips and Tricks in SELECT Statements



Copyright © Capgemini 2016. All Rights Reserved



Table of Contents

- Lesson 4: Aggregate (Group) functions
 - 4.1: The Group function
 - 4.2: GROUP BY & HAVING clause
 - 4.3: Examples of GROUP BY and HAVING clauses
 - 4.4: Tips and Tricks
- Lesson 5: SQL (Single-row) functions
 - 5.1: SQL functions
 - 5.2: Number functions
 - 5.3: Character functions



Copyright © Capgemini 2016. All Rights Reserved 10

Table of Contents

- Lesson 5: SQL (Single-row) functions (contd.)
 - 5.4: Date functions
 - 5.5: Conversion functions
 - 5.6: Miscellaneous functions
 - 5.7: Tips and Tricks
- Lesson 6: Joins and Sub-queries
 - 6.1: Joins
 - 6.1.1: Oracle Proprietary Joins
 - 6.1.2: SQL: 1999 Compliant Joins



Copyright © Capgemini 2016. All Rights Reserved. 11

Table of Contents

- Lesson 6: Joins and Sub-queries (contd.)
 - 6.6: Sub-queries
 - 6.7: Co-related sub-query
 - 6.8: Exists / Not Exists Operator
 - 6.9: CONNECT BY and START WITH clauses
 - 6.10: Tips and Tricks
- Lesson 7: Set Operators
 - 7.1: Set Operation
 - 7.2: The UNION Operator
 - 7.3: The INTERSECT Operator
 - 7.4: The MINUS Operator



Copyright © Capgemini 2016. All Rights Reserved 12

Table of Contents

- Lesson 8: Database Objects
 - 8.1: Basic Data Types
 - 8.2: Data Integrity
 - 8.3: Examples of CREATE TABLE
 - 8.4: Examples of ALTER TABLE
 - 8.5: Database Objects
 - 8.6: Index
 - 8.7: Synonym
 - 8.8: Sequence
 - 8.9: View
 - 8.10: Deleting Database Objects



Copyright © Capgemini 2016. All Rights Reserved 13

Table of Contents

- Lesson 9: Data Manipulation Language
 - 9.1: Adding Data
 - 9.2: Removing Data
 - 9.3: Modifying Data
- Lesson 10: Transaction Control Language
 - 10.1: Introduction to Transactions
 - 10.2: Statement Execution and Transaction Control
- Lesson 11: Transaction Control Language
 - 11.1: Object Privileges
 - 11.2: Grant and Revoke Statements



Copyright © Capgemini 2016. All Rights Reserved. 54

References

- RDBMS Concepts - A Primer
 - <http://safari.oreilly.com>
- Introduction to Database Systems; by C.J.Date
- Relational Database Theory; by Atzeni, De Antonellis



Next Step Courses (if applicable)

- Oracle PL/SQL



Copyright © Capgemini 2016. All Rights Reserved 16

DBMS/SQL

Lesson 01: Getting Started
with Database

Lesson Objectives

- To understand the following topics:
 - Database
 - Features of DBMS
 - DBMS models
 - Relational DBMS



1.1: Introduction to Database

What is Data?

- Data (plural of the word datum) is a factual information used as a basis for reasoning, discussion, or calculation
- Data may be numerical data which may be integers or floating point numbers, and non-numerical data such as characters, date etc.
- Data by itself normally doesn't have a meaning associated with it.

e.g:- Jack
 01-jan-71
 15-jun-05
 50000



Copyright © Capgemini 2015. All Rights Reserved. 3

1.1: Introduction to Database

What is Information?

- Related data is called as information
- Information will always have a meaning and context attached to the data element
- When we add meaning and context to the data it becomes information.
 - Employee name: Jack
 - Date of birth: 01-jan-71
 - Data of joining: 15-jun-05
 - Salary: 50000
 - Department number: 10

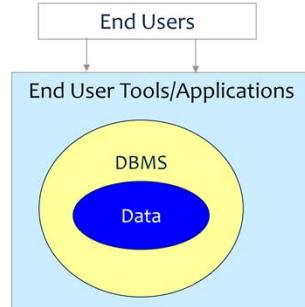


Copyright © Capgemini 2015. All Rights Reserved. 4

1.1: Introduction to Database

Defining Database, DBMS & Schema

- Database: It is a set of inter-related data
- DBMS: It is a software that manages the data
- Schema: It is a set of structures and relationships, which meet a specific need



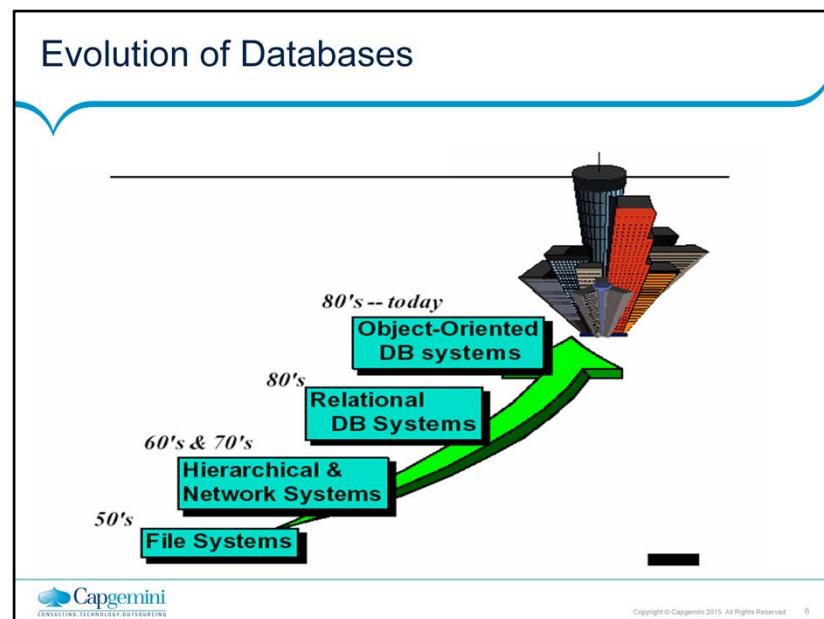
Copyright © Capgemini 2015. All Rights Reserved 5

Introduction to Database:

A logically coherent collection of related data ("information") with inherent meaning, built for a certain application, and representing a subset of the "real-world". For eg: Customer database in bank, Employee Details

The software that manages the database is known as "Database Management System" or "DBMS". Hence DBMS can be described as "a computer-based record keeping system which consists of software for processing a collection of interrelated data". The general purpose of a DBMS is to provide for the definition, storage, and management of data in a centralized area that can be shared by many users

A set of structures and relationships that meet a specific need is called as a "schema".



1.2: Features of DBMS

Characteristics of DBMS

- Given below are the characteristics of DBMS:
- Control of Data Redundancy
 - Traditionally, same data is stored in a number of places
 - Gives rise to data redundancy and its disadvantages
 - DBMS helps in removing data redundancies by providing means of data- integration.
- Sharing of Data
 - DBMS allows many applications to share the data.
- Maintenance of Integrity
 - DBMS maintains the correctness, consistency, and interrelationship of data with respect to the application, which uses the data.



Copyright © Capgemini 2015. All Rights Reserved 7

Characteristics of DBMS:

Some of the characteristics of the DBMS are given below:

Control of Data Redundancy

When the same data is stored in a number of files, it results in data redundancy. In such cases, if the data is changed at one place, the change has to be duplicated in each of the files.

The main disadvantages of data redundancy are:

Storage space is wasted.

Processing time may be wasted as more data needs to be handled.

Inconsistencies may creep in.

DBMS helps in removing redundancies by providing means of integration.

Sharing of Data

DBMS allows many applications to share the data.

Maintenance of Integrity

Integrity of data refers to the correctness, consistency and interrelationship of data with respect to the application that uses the data. Some of the aspects of data integrity are:

Many data items can only take a restricted set of values.

Characteristics of DBMS

- Support for Transaction Control and Recovery
 - DBMS ensures that updates physically take place after a logical Transaction is complete.
- Data Independence
 - In DBMS, the application programs are transparent to the physical organization and access techniques.
- Availability of Productivity Tools
 - Tools like query language, screen and report painter, and other 4GL tools are available.



Copyright © Capgemini 2015. All Rights Reserved. 8

Characteristics of DBMS (contd.):

Certain field values cannot be duplicated across records. Such restrictions, called primary key constraints, can be defined to the DBMS.

Data integrity, which defines the relationships between different files, is called referential integrity rule, which can also be specified to the DBMS

Support for Transaction Control and Recovery

Multiple changes to the database can be clubbed together as a single "logical transaction".

The DBMS ensures that the updates take place physically, only when the logical transaction is complete.

Data Independence

In conventional file based applications, programs need to know the "data organization" and "access technique" to be able to access the data.

This means that if you make any change in the manner the data is organized, then you have to make changes to the application programs that apply to the data.

In DBMS, the application programs are transparent to the "physical organization" and "access techniques".

Characteristics of DBMS

- Control over Security
 - DBMS provides tools with which the DBA can ensure security of the database.
- Hardware Independence
 - Most DBMS are available across hardware platforms and operating systems.



Copyright © Capgemini 2015. All Rights Reserved 9

Characteristics of DBMS (contd.):

Availability of Productivity Tools

Tools like query language, screen and report painter, and other 4GL tools are available.

These tools can be utilized by the end-users to query, print reports, etc. SQL is one such language, which has emerged as standard.

Security

DBMSes provide tools, which can be used by the DBA to ensure security of the database.

Hardware Independence

Most DBMSes are available across hardware platforms and operating systems.

Thus the application programs need not be changed or rewritten when the “hardware platform” or “operating system” is changed or upgraded.

Levels of Abstraction

- There are three levels of database abstraction:
 - Conceptual Level:
 - The overall integrated structural organization of the database.
 - Physical Level:
 - The information about how the database is actually stored in the disk.
 - View / External Level:
 - The user view of the database. It is different for different users based on application requirement.



Copyright © Capgemini 2015. All Rights Reserved 10

1.3: The Data Models

What is a Data Model?

- The “Data model” defines the range of data structures supported and the availability of data handling languages.
- It is a collection of conceptual tools to describe:
 - Data
 - Data relationships
 - Constraints
- There are different data models:
 - Hierarchical Model
 - Network Model
 - Relational Model



Copyright © Capgemini 2015. All Rights Reserved 11

What is a Data Model?

A “Data model” is a conceptual representation of the data structures that are required by a database. The data structures include:

- the data objects
- the associations between data objects, and
- the rules which govern operations on the objects

As the name implies, the “Data model” focuses on the data that is required, and how it should be organized rather than the operations that will be performed on the data.

The DBMS MODELS

The range of “data structures” that are supported, and the availability of data handling languages depend on the model of DBMS on which it is based. The models are:

- The hierarchical model
- The network model
- The relational model

Why is Data Modeling Important?

- Why is Data Modeling important?
 - The goal of the “data model” is to ensure that all the data objects required by the database are completely and accurately represented.
 - The “data model” uses easily understood notations and natural language. Hence, it can be reviewed and verified as correct by the end-users.



Copyright © Capgemini 2015. All Rights Reserved. 12

Why is Data Modeling Important?

The “data model” is also detailed enough to be used, by the database developers, as a “blueprint” for building the physical databases. The information contained in the “data model” will be used to define the relational tables, primary and foreign keys, stored procedures, and triggers.

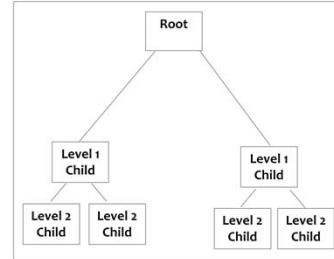
Poorly designed databases require more time in the long-term. Without careful planning you may create a database that:

- Omits data required to create critical reports.
- Produces results that are incorrect or inconsistent.
- Is unable to accommodate changes in the user requirements.

1.3: The Data Models

Hierarchical Model

- The Hierarchical model:
- In this model, data is represented by a simple tree-structure.
- Relationships between entities are represented as parent-child.
- Many-to-many relationships are not allowed.
- Parents and children are tied together by links called "pointers".



Hierarchical Model- Example

Example:

- Consider a student course - marks database.
- In the Hierarchical model a student can register for many courses and gets marks for each course.

- A parent can have many children
- A child cannot have more than one parent
- No child can exist without its parent

Scode	Sname
S1	A

Scode	Sname
S2	B

Ccode	Cname	Marks
C1	Physics	65
C2	Chemistry	78
C3	Maths	83
C4	Biology	85

Ccode	Cname	Marks
C3	Maths	83
C4	Biology	85



Copyright © Capgemini 2015. All Rights Reserved 14

Example of a Hierarchical model:

Consider a student course - marks database. In the Hierarchical model a student can register for many courses, and get marks for each course.

The student record is called as "root". It has got a course - marks record that is called as "child record".

In general:

- A parent can have many children.
- A child cannot have more than one parent.
- No child can exist without its parent.

Hierarchical Model - Possibilities

- Possibilities in a Hierarchical model:
- **INSERT**
 - Insertion of Dummy student is required to introduce a new course.
- **DELETE**
 - Deleting a student - the only one to take the course deletes course information.
- **UPDATE**
 - To change the course name of one course, the whole database has to be searched. This may result in data inconsistency.



Copyright © Capgemini 2015. All Rights Reserved 15

Possibilities in a Hierarchical model:

In the Hierarchical model, following possibilities exist:

INSERT

Since no child record can exist without its parent, it is not possible to insert the new course details without introducing a dummy student record.

DELETE

If a course is selected by only one student, then deleting that student will automatically delete all information about the course.

UPDATE

To change the course name of one course, the whole database has to be searched. This may result in data inconsistency.

1.3: The Data Models

Network Model

- The Network model:

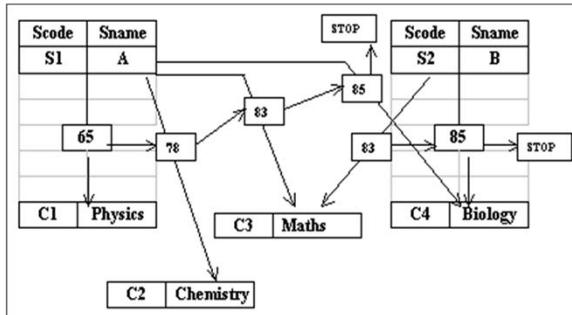
- The Network model solves the problem of data redundancy by representing relationships in terms of "sets" rather than "hierarchy".
- A record occurrence may have any number of immediate superiors.
- The Network model supports many-to-many relationships.
- There is no restriction on number of parents.
- A record type can have a number of parent and child record types.
- It is more complex than the Hierarchical model because of links.
- It is a superset of the Hierarchical model.



Copyright © Capgemini 2015. All Rights Reserved 16

Network Model - Example

- In the example of student course – marks, “student record” and “course record” is linked together through “marks record”.



Copyright © Capgemini 2015. All Rights Reserved 17

Example of Network model:

In the Network model, the “student record” and “course record” is linked together through a “marks record”.

There are no restrictions on number of parents.

A record type can have any number of “parent” and “child” record types.

The Network model is more complex than the Hierarchical model because of it's links.

The Network model can represent any structure that is designed in the Hierarchical model. Hence, it is a superset of the Hierarchical model. a

Network Model - Possibilities

- Possibilities in a Network model:
 - INSERT
 - Inserting a "course record" or "student record" poses no problems. They can exist without any connectors till a student takes the course.
 - DELETE
 - Deleting any record automatically adjusts the chain.
 - UPDATE
 - Update can be done only to a particular child record.



Copyright © Capgemini 2015. All Rights Reserved 18

Possibilities in a Network model:

In Network model, following possibilities exist:

INSERT

Inserting a course record or student record poses no problems, as they can exist without any connectors till a student takes the course.

DELETE

Deleting any record automatically adjusts the chain.

UPDATE

Update can be done only to a particular child record.

1.3: The Data Models

Relational Model

- The Relational model:
 - The Relational model developed out of the work done by Dr. E. F. Codd at IBM in the late 1960s. He was looking for ways to solve the problems with the existing models.
 - At the core of the Relational model is the concept of a “table” (also called a “relation”), which stores all data.
 - Each “table” is made up of:
 - “records” (i.e. horizontal rows that are also known as “tuples”), and
 - “fields” (i.e. vertical columns that are also known as “attributes”)



Copyright © Capgemini 2015. All Rights Reserved 19

Relational Model

- The Relational model:
- Examples of RDBMS:
 - Oracle
 - Informix
 - Sybase
- Because of lack of linkages, the Relational model is easier to understand and implement.

Student Table	
Ccode	Sname
S1	A
S2	B

Course Table	
Ccode	Cname
C1	Physics
C2	Chemistry
C3	Maths
C4	Biology

Marks Table		
Ccode	Scode	Marks
C1	S1	65
C2	S1	78
C3	S1	83
C4	S1	85
C3	S2	83
C4	S2	85



Copyright © Capgemini 2015. All Rights Reserved 20

Relational Model - Possibilities

- Possibilities in a Relational model:
- **INSERT**
 - Inserting a "course record" or "student record" poses no problems because tables are separate.
- **DELETE**
 - Deleting any record affects only a particular table.
- **UPDATE**
 - Update can be done only to a particular table.



Copyright © Capgemini 2015. All Rights Reserved 21

1.4: Relational DBMS

Relational Tables

Examples of Relational tables:

Dept table				
Deptno	Dname	Loc	"column" or "attribute"	
10	Accounting	New York		
20	Research	Dallas	"row" or "tuple"	
30	Sales	Chicago		
40	Operations	Boston		

Emp table				
Empno	Empname	Job	Mgr	Deptno
7369	Smith	Clerk	7902	20
7499	Allen	Salesman	7839	30
7566	Jones	Manager	7839	20
7839	King	President		10
7902	Ford	Analyst	7566	20



Copyright © Capgemini 2015. All Rights Reserved 22

Relational DBMS (RDBMS):

The Relational model presents an orderly, predictable, and intuitive approach for:

- Organizing data,
- Manipulating data, and
- Viewing data

RDBMS Terminology

Relational data consists of relations.

A relation (or relational table) is a “two dimensional” table with special properties.

A relational table consists of:

- a set of named columns, and
- an arbitrary number of rows

The columns are called as “attributes” or “fields”. The rows are called as “tuples” or “records”.

Each “attribute” is associated with a “domain”.

A “domain” is a set of values that may appear in one or more columns.

Relational Tables - Properties

- Properties of Relational Data Entities:
 - Tables must satisfy the following properties to be classified as relational:
 - Entries of attributes should be single-valued.
 - Entries of attributes should be of the same kind.
 - No two rows should be identical.
 - The order of attributes is unimportant.
 - The order of rows is unimportant.
 - Every column can be uniquely identified.



Copyright © Capgemini 2015. All Rights Reserved 23

Properties of Relational Data Entities:

Relational tables have six properties, which must be satisfied for any table to be classified as Relational. These are :

1.4: Relational DBMS

Data Integrity

- “Data Integrity” is the assurance that data is consistent, correct, and accessible throughout the database.
- Some of the important types of integrities are:
 - Entity Integrity:
 - It ensures that no “records” are duplicated, and that no “attributes” that make up the primary key are NULL.
 - It is one of the properties that is necessary to ensure the consistency of the database.



Copyright © Capgemini 2015. All Rights Reserved 24

Data Integrity:

Data Integrity refers to the wholeness and soundness of the database. Some of the most important integrities are given below.

Domain Constraints

A “domain” is a set of values that are permitted to appear in one or more columns. Once a “domain” is specified and a “column” is associated with the “domain”, then the “column” can take only those values that are permitted by the “domain”.

Primary Key and Entity Integrity

“Primary key” is a “column” or “set of columns” in a table which uniquely identifies a “row” in a table.

No two rows of the table can have the same values for the Primary key.

Entity integrity is maintained by ensuring that none of the columns that make up the Primary key can take “NULL” (unknown) values.

Data Integrity

- Foreign Key and Referential Integrity
 - The Referential Integrity rule: If a Foreign key in table A refers to the Primary key in table B, then every value of the Foreign key in table A must be null or must be available in table B.
- Unique Constraint:
 - It is a single field or combination of fields that uniquely defines a tuple or row.
 - It ensures that every value in the specified key is unique.
 - A table can have any number of unique constraints, with at most one unique constraint defined as a Primary key.
 - A unique constraint can contain NULL value.



Copyright © Capgemini 2015. All Rights Reserved 25

Data Integrity:

Data Integrity refers to the wholeness and soundness of the database. Some of the most important integrities are given below.

Domain Constraints

A “domain” is a set of values that are permitted to appear in one or more columns. Once a “domain” is specified and a “column” is associated with the “domain”, then the “column” can take only those values that are permitted by the “domain”.

Primary Key and Entity Integrity

“Primary key” is a “column” or “set of columns” in a table which uniquely identifies a “row” in a table.

No two rows of the table can have the same values for the Primary key.

Entity integrity is maintained by ensuring that none of the columns that make up the Primary key can take “NULL” (unknown) values.

Data Integrity (contd.):

Update Cascade referential Integrity

This means that if a Primary key value is updated, then all Foreign key values dependent on it will be updated to the new value of Primary key. That is to say, if we change the deptno 10 to 50, then all the employees in deptno 10 will be shifted, as well, to deptno 50 (column deptno will be automatically updated).

Column Constraints

These are the constraints, which specify restrictions on the values that can be taken by a column. These restrictions may be defined with or without other values in the same row.

Data Integrity

- Column Constraint:
 - It specifies restrictions on the values that can be taken by a column.

DEPT table		
Deptno	Dname	Loc
10	Accounting	New York
20	Research	Dallas

EMP table				
Empno	Empname	Job	Mgr	Deptno
7369	Smith	Clerk	7902	20
7499	Allen	Salesman	7839	30



Copyright © Capgemini 2015. All Rights Reserved 27

Database Administrator

- A Database Administrator (DBA) is the database architect.
- DBA is responsible for the design and implementation of new databases, and:
 - centrally manages the database.
 - decides on the type of data, internal structures, and their relationships
 - ensures the security of the database
 - controls access to the data through user codes and passwords
 - can restrict the views or operations that the users can perform on the database



Copyright © Capgemini 2015. All Rights Reserved 28

Data Integrity (contd.):

Update Cascade referential Integrity

This means that if a Primary key value is updated, then all Foreign key values dependent on it will be updated to the new value of Primary key. That is to say, if we change the deptno 10 to 50, then all the employees in deptno 10 will be shifted, as well, to deptno 50 (column deptno will be automatically updated).

Column Constraints

These are the constraints, which specify restrictions on the values that can be taken by a column. These restrictions may be defined with or without other values in the same row.

Summary

- In this lesson, you have learnt:
 - What is a Database?
 - Characteristics of DBMS
 - The Data models, including:
 - the Hierarchical model
 - the Network model
 - the Relational model
 - Relational DBMS (RDBMS)
 - What is Data Integrity?



Review Question

- Question 1: A DBA ____.
 - Option 1: ensures the security of the application server.
 - Option 2: controls access to the data through the user codes and passwords.
 - Option 3: manages users.

- Question 2: The Physical Level is ____.
 - Option 1: the overall structural organization of the d/b.
 - Option 2: the information about how the database is actually stored in the disk.
 - Option 3: the user view of the database.



Copyright © Capgemini 2015. All Rights Reserved 30

Review Question

- Question 3: There are different data models such as ____.
- Question 4: In Network model, each table is made up “tuples” and “fields”.
 - True / False
- Question 5: A table can have any number of “Unique constraints”.
 - True / False



Copyright © Capgemini 2015. All Rights Reserved. 31

DBMS SQL

Lesson 2: Basics of SQL

Lesson Objectives

- To understand the following topics:
 - SQL, rules for SQL Statements, standard SQL Statement groups
 - SQL*PLUS & iSQLPlus environment



2.1: SQL

What is SQL?

- SQL:
 - SQL stands for Structured Query Language.
 - SQL is used to communicate with a database.
 - Statements are used to perform tasks such as update data on a database, or retrieve data from a database.
 - Benefits of SQL are:
 - It is a Non-Procedural Language.
 - It is a language for all users.
 - It is a unified language.

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 3

What is SQL?

SQL stands for Structured Query Language. Pronounced as SEQUEL, SQL was

originally developed by IBM based on Dr. E.F. Codd's relational model to

define, manipulate and control data in a database

SQL is a set of commands that allows you to access a Relational Database.

SQL is an ANSI standard computer language.

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.

SQL is a non-procedural, English-like language that processes data in "groups of records" rather than processing one record at a time.

SQL provides automatic navigation to the data.

What can SQL do?

- SQL
 - allows you to access a database.
 - can execute queries against a database.
 - can retrieve data from a database.
 - can insert new records into a database.
 - can delete records from a database.
 - can update records in a database.



Copyright © Capgemini 2015. All Rights Reserved. 4

Almost all modern Relational Database Management Systems like MS SQL Server, Microsoft Access, MSDE, Oracle, DB2, Sybase, MySQL, Postgres, and Informix use SQL as standard database language.

However, although all those RDBMS use SQL, they use different SQL dialects

For example: MS SQL Server specific version of the SQL is called T-SQL, Oracle version of SQL is called PL/SQL which also supports Procedural Language features, MS Access version of SQL is called JET SQL, etc.

Oracle has further modified SQL to support ORDBMS features.

2.1: SQL

Rules for SQL statements

- Rules for SQL statements:
 - SQL keywords are not case sensitive. However, normally all commands (SELECT, UPDATE, etc) are upper-cased.
 - “Variable” and “parameter” names are displayed as lower-case.
 - New-line characters are ignored in SQL.
 - Many DBMS systems terminate SQL statements with a semi-colon character.
 - “Character strings” and “date values” are enclosed in single quotation marks while using them in WHERE clause or otherwise.

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 5

Add the notes here.

2.1: SQL

Standard SQL statement groups

Given below are the standard SQL statement groups:

Groups	Statements	Description
DQL	SELECT	DATA QUERY LANGUAGE – It is used to get data from the database and impose ordering upon it.
DML	DELETE INSERT UPDATE MERGE	DATA MANIPULATION LANGUAGE – It is used to change database data.
DDL	DROP TRUNCATE CREATE ALTER	DATA DEFINITION LANGUAGE – It is used to manipulate database structures and definitions.
TCL	COMMIT ROLLBACK SAVEPOINT	TCL statements are used to manage the transactions.
DCL (Rights)	REVOKE GRANT	They are used to remove and provide access rights to database objects.

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 6

Standard SQL Statement groups:

“SQL commands” are “instructions” used to communicate with the database to perform “specific tasks” that work with data.

A database is a collection of structures with appropriately defined authorizations and accesses. The tables, indexes are structures in the database and are called as “objects” in the database.

The names of tables, indexes, and those of columns are called “identifiers”.

SQL commands can be used not only for searching the database, but also to perform various other functions.

For example: You can create tables, add data to tables, or modify data, drop the table, set permissions for users, etc.

Standard SQL Statement groups:

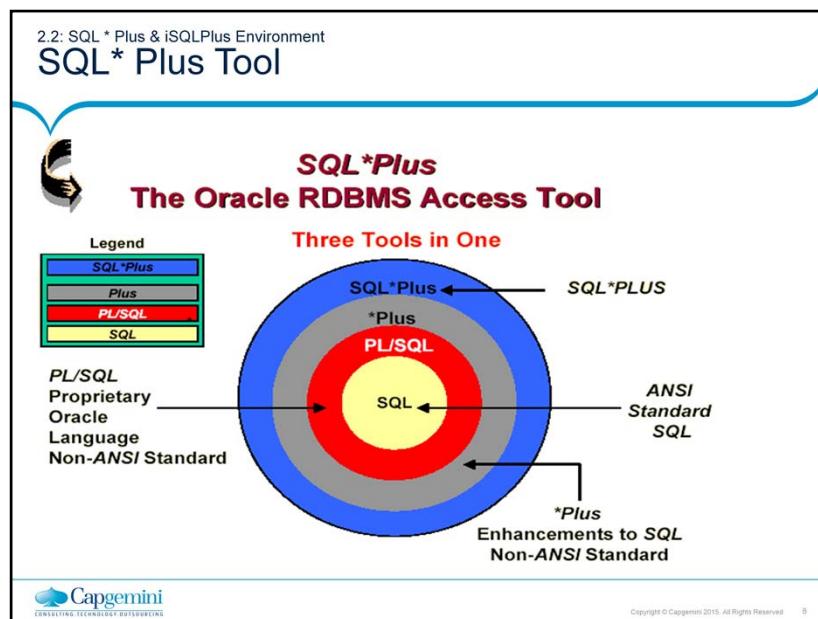
“SQL commands” are “instructions” used to communicate with the database to perform “specific tasks” that work with data.

A database is a collection of structures with appropriately defined authorizations and accesses. The tables, indexes are structures in the database and are called as “objects” in the database.

The names of tables, indexes, and those of columns are called “identifiers”.

SQL commands can be used not only for searching the database, but also to perform various other functions.

For example: You can create tables, add data to tables, or modify data, drop the table, set permissions for users, etc.



SQL*PLUS:

SQL* PLUS is an Oracle tool which accepts SQL, and SQLPLUS commands and PL/SQL blocks, and executes them. SQL*Plus commands can be abbreviated

2.2: SQL * Plus & iSQLPlus Environment

Logging to Oracle Server using iSQLPlus

- To log into the iSQL*Plus environment:
- In the Windows browser, type the URL in the address field. The user will be directed to iSQL*Plus environment screen.



The screenshot shows a web browser window with the URL <http://rgxxxx1.ap1.oracle.com/rsqlplus> in the address bar. The page title is "iSQL*Plus". It contains four input fields: "Username", "Password", "Connection Identifier", and a dropdown menu for "Privilege" set to "User". Below the fields are "Log In" and "Clear" buttons. The browser toolbar at the top includes Back, Forward, Reload, Home, Search, Netscape, Print, Security, Stop, and Help buttons. A Capgemini logo is visible at the bottom left, and a copyright notice at the bottom right.

Oracle provides a browser based facility also to access SQL* Plus i.e iSQLPlus. Unlike SQL*Plus the commands in iSQLPlus cannot be abbreviated. Both SQL*Plus and iSQLPLus are Oracle proprietary interfaces.

How to access SQL* PLUS ?

Get your Oracle user name and password, and connect string from your DBA or Instructor.

As shown on the slide you get the login screen wherein the required details can be put in to login into Oracle

2.2: SQL * Plus & iSQLPlus Environment

Logging to Oracle Server using iSQLPlus

- To connect to the Oracle server:
 - Select Start, go to Programs, and select Oracle-OraHome8i.
 - Go to Application Development, and select SQL Plus. You will get the following logon screen:



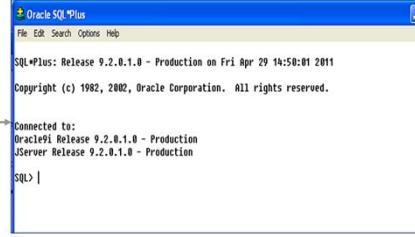
Log On

User Name:

Password:

Host String:

OK Cancel



Oracle SQL*Plus

File Edit Search Options Help

SQL*Plus: Release 9.2.0.1.0 - Production on Fri Apr 29 14:50:01 2011

Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

Connected to:

Oracle9i Release 9.2.0.1.0 - Production

JavaServer Release 9.2.0.1.0 - Production

SQL> |

Connect to SQL*Plus

Copyright © Capgemini 2015. All Rights Reserved 10

Similarly you can use the username, password and Host String details to use the SQL * Plus tool. This is a GUI based tool to interact with Oracle

To enter a SQL> command, type the command at the SQL > prompt. An SQL command can be continuously entered on a single line, or it can be spread over multiple lines. In case of any typing error, use BACKSPACE or DEL key to erase, and then continue typing.

```
SQL> SELECT empno,hiredate FROM emp  
WHERE hiredate >  
      '01-jan-82' ;
```

The command can also be entered as:

```
SQL>SELECT empno,hiredate  
FROM emp
```

```
WHERE hiredate >'01-jan-82';
```

Summary

- In this lesson, you have learnt:
- What is SQL?
 - Rules for SQL statements
 - Standard SQL statement groups
- SQL*PLUS environment



Review Question

- Question 1: SQL ____.
 - Option 1: cannot execute queries against a database.
 - Option 2: can manipulate data from a database.
 - Option 3: cannot retrieve data from a database.
 - Option 4: can insert new records in a database.
 - Option 5: can delete records from a database.



Review Question

- Question 2: SQL categories are ____.
 - Option 1: DDL
 - Option 2: DML
 - Option 3: DSL
 - Option 4: DQL
 - Option 5: TCL
 - Option 6: TDL



DBMS/SQL

Lesson 03: Data Query Language
(The Select Statement)

Lesson Objectives

- To understand the following topics:
 - The SELECT statement
 - The WHERE clause
 - The DISTINCT clause
 - The Comparison, Arithmetic, and Logical operators
 - The ORDER BY clause



3.1: The SELECT Statement

The Select Statement and Syntax

- The SELECT command is used to retrieve rows from a single table or multiple Tables or Views.
- A query may retrieve information from specified columns or from all of the columns in the Table.
- It helps to select the required data from the table.

```
SELECT [ALL | DISTINCT] { * | col_name,... }
FROM table_name alias, ...
[ WHERE expr1 ]
[ CONNECT BY expr2 [ START WITH expr3 ] ]
[ GROUP BY expr4 ] [ HAVING expr5 ]
[ UNION | INTERSECT | MINUS SELECT ... ]
[ ORDER BY expr | ASC | DESC ];
```



Copyright © Capgemini 2015. All Rights Reserved. 3

The SELECT Statement:

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set). The statement begins with the SELECT keyword. The basic SELECT statement has three clauses:

```
SELECT
  FROM
  WHERE
```

The SELECT clause specifies the table columns that are retrieved.

The FROM clause specifies the tables accessed.

The WHERE clause specifies which table rows are used. The WHERE clause is optional; if missing, all table rows are used.

Note:

Each clause is evaluated on the result set of a previous clause. The final result of the query will be always a "result table".

Only FROM clause is essential. The clauses WHERE, GROUP BY, HAVING, ORDER BY, UNION are optional.

All the examples that follow are based on EMP and DEPT tables that are already available.

Selecting Columns

- Displays all the columns from the student_master table

```
SELECT * FROM student_master;
```

- Displays selected columns from the student_master table

```
SELECT student_code,  
       student_name  
    FROM student_master;
```



Copyright © Capgemini 2015. All Rights Reserved. 4

3.2: SELECT statement Clauses

The WHERE clause

- The WHERE clause is used to specify the criteria for selection.
- For example: displays the selected columns from the student_master table based on the condition being satisfied

```
SELECT student_code, student_name, student_dob  
      FROM student_master  
     WHERE dept_code = 10;
```



Copyright © Capgemini 2015. All Rights Reserved. 5

The WHERE Clause:

The WHERE clause is used to perform “selective retrieval” of rows. It follows the FROM clause, and specifies the search condition.

The result of the WHERE clause is the row or rows retrieved from the Tables, which meet the search condition.

The clause is of the form:

Comparison Predicates:

The Comparison Predicates specify the comparison of two values.

It is of the form:

< Expression > < operator > <
Expression>
< Expression > <operator>
<subquery>

The operators used are shown on the next slide:

```
WHERE <search condition>
```

contd.

3.2: SELECT statement Clauses

Character Strings and Dates

- Are enclosed in single quotation marks
- Character values are case sensitive
- Date values are format sensitive

```
SELECT student_code, student_dob  
      FROM student_master  
 WHERE student_name = 'Sunil' ;
```



Copyright © Capgemini 2015. All Rights Reserved. 6

Oracle Database store dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds.

The date datatype is covered in detail later.

3.2: SELECT statement Clauses

Mathematical, Comparison & Logical Operators

■ Mathematical Operators:

- Examples: +, -, *, /

■ Comparison Operators:

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or Equal to
<	Less than
<=	Less than or Equal to
<>, !=, or ^=	Not Equal to

■ Logical Operators:

- Examples: AND, OR, NOT



Copyright © Capgemini 2015. All Rights Reserved. 7

Operators:

Operators are used in “expressions” or “conditional statements”. They show equality, inequality, or a combination of both.

Operators are of three types:

- mathematical
- logical
- range (comparison)

These operators are mainly used in the WHERE clause, HAVING clause in order to filter the data to be selected.

Mathematical operators:

These operators add, subtract, multiply, divide, and compare equality of numbers and strings. They are +, -, *, /

Comparison Operators:

These operators are used to compare the column data with specific values in a condition. “Comparison Operators” are also used along with the “SELECT statement” to filter data based on specific conditions. The table in the slide describes each Comparison operator. Comparison operators indicate how the data should relate to the given search value.

Logical Operators:

There are three Logical Operators namely AND, OR and NOT. These operators compare two conditions at a time to determine whether a row can be selected for the output or not. When retrieving data by using a SELECT statement, you can use logical operators in the WHERE clause. This allows you to combine more than one condition.

3.2: SELECT statement Clauses

Other Comparison Operators

Other Comparison operators	Description
[NOT] BETWEEN x AND y	Allows user to express a range. For example: Searching for numbers BETWEEN 5 and 10. The optional NOT would be used when searching for numbers that are NOT BETWEEN 5 AND 10.
[NOT] IN(x,y,...)	Is similar to the OR logical operator. Can search for records which meet at least one condition contained within the parentheses. For example: Pubid IN (1, 4, 5), only books with a publisher id of 1, 4, or 5 will be returned. The optional NOT keyword instructs Oracle to return books not published by Publisher 1, 4, or 5.



Copyright © Capgemini 2015. All Rights Reserved. 8

3.2: SELECT statement Clauses

Other Comparison Operators

Other Comparison operators	Description
[NOT] LIKE	Can be used when searching for patterns if you are not certain how something is spelt. For example: title LIKE 'TH%'. Using the optional NOT indicates that records that do contain the specified pattern should not be included in the results.
IS[NOT]NULL	Allows user to search for records which do not have an entry in the specified field. For example: Shipdate IS NULL. If you include the optional NOT, it would find the records that do not have an entry in the field. For example: Shipdate IS NOT NULL.

3.2: SELECT statement Clauses

BETWEEN ... AND Operator

- The BETWEEN ... AND operator finds values in a specified range:

```
SELECT staff_code,staff_name FROM staff_master  
WHERE staff_dob  
      BETWEEN '01-Jan-1980'  
      AND '31-Jan-1980';
```



Copyright © Capgemini 2015. All Rights Reserved. 10

3.2: SELECT statement Clauses

IN Operator

- The IN operator matches a value in a specified list.
 - The List must be in parentheses.
 - The Values must be separated by commas.

```
SELECT dept_code FROM department_master  
WHERE dept_name IN ('Computer Science', 'Mechanics');
```



Copyright © Capgemini 2015. All Rights Reserved. 11

IN predicate:

It is of the form:

<Expression> IN <LIST>
<Expression> IN <SUBQUERY>

The data types should match.

3.2: SELECT statement Clauses

IN Operator

- The IN operator matches a value in a specified list.
 - The List must be in parentheses.
 - The Values must be separated by commas.

```
SELECT dept_code FROM department_master  
WHERE dept_name IN ('Computer Science', 'Mechanics');
```



Copyright © Capgemini 2015. All Rights Reserved. 12

IN predicate:

It is of the form:

<Expression> IN <LIST>
<Expression> IN <SUBQUERY>

The data types should match.

3.2: SELECT statement Clauses

LIKE Operator

- The LIKE operator performs pattern searches.
 - The LIKE operator is used with wildcard characters.
 - Underscore (_) for exactly one character in the indicated position
 - Percent sign (%) to represent any number of characters

```
SELECT book_code,book_name FROM book_master  
WHERE book_pub_author LIKE '%Kanetkar%';
```



CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 13

LIKE predicate:

It is of the form:

<COLUMN > LIKE < PATTERN>

The pattern contains a search string along with other special characters % and _. The % character represents a string of any length where as _ (underscore) represents exactly one character. A pattern %XYZ% means search has to be made for string XYZ in any position. A pattern '_XYZ%' means search has to be made for string XYZ in position 2 to 4.

To search for characters % and _ in the string itself we have to use an "escape" character.

For example: To search for string NOT_APP in column status, we have to use the form Status like 'NOT_APP' ESCAPE '\'

The use of \ as escape character is purely arbitrary.

3.2: SELECT statement Clauses

Logical Operators

- Logical operators are used to combine conditions.
- Logical operators are NOT, AND, OR.
 - NOT reverses meaning.
 - AND both conditions must be true.
 - OR at least one condition must be true.
- Use of AND operator

```
SELECT staff_code,staff_name,staff_sal FROM staff_master  
WHERE dept_code = 10  
AND staff_dob > '01-Jan-1945';
```



Copyright © Capgemini 2015. All Rights Reserved. 14

The AND operator displays a record if both the first condition and the second condition is true.

One More Example:

```
SQL>SELECT title, pubid, category  
2   FROM books  
3 WHERE pubid = 3  
4 AND category = 'COMPUTER';
```

Combining Predicates by using Logical Operators:

The predicates can be combined by using logical operators like AND, OR, NOT. The evaluation proceeds from left to right and order of evaluation is:

- * Enclosed in parenthesis
- AND
- OR

3.2: SELECT statement Clauses

Using AND or OR Clause

- Use of OR operator:

```
SELECT book_code FROM book_master  
WHERE book_pub_author LIKE '%Kanetkar%'  
OR book_name LIKE '%Pointers%';
```



Copyright © Capgemini 2015. All Rights Reserved. 15

The OR operator displays a record if either the first condition or the second condition is true.

You can also combine AND and OR as shown in above example.
(use parenthesis to form complex expressions).

3.2: SELECT statement Clauses

Using NOT Clause

- The NOT operator finds rows that do not satisfy a condition.
- For example: List staff members working in depts other than 10 & 20.

```
SELECT staff_code,staff_name FROM staff_master  
WHERE dept_code NOT IN ( 10,20 );
```



Copyright © Capgemini 2015. All Rights Reserved. 16

Note: NOT is a negation operator.

3.2: SELECT statement Clauses

Treatment of NULL Values

- NULL is the absence of data.
- Treatment of this scenario requires use of IS NULL operator.

```
SQL>      SELECT student_code  
2       FROM student_master  
3       WHERE dept_code IS NULL;
```



Copyright © Capgemini 2015. All Rights Reserved. 17

NULL predicate:

The NULL predicate specifies a test for NULL values. The form for NULL predicate is:

< COLUMN SPECIFICATION > IS NULL.
< COLUMN SPECIFICATION > IS NOT NULL.
< COLUMN SPECIFICATION > IS NULL returns

TRUE only when column has NULL values.

<COLUMN> = NULL cannot be used to compare null values.

Operator Precedence

Operator precedence is decided in the following order:

Levels	Operators
1	* (Multiply), / (Division), % (Modulo)
2	+ (Positive), - (Negative), + (Add), (+ Concatenate), - (Subtract), & (Bitwise AND)
3	=, >, <, >=, <=, !>, !< (Comparison operators)
4	NOT
5	OR
6	AND
7	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
8	= (Assignment)



Copyright © Capgemini 2015. All Rights Reserved. 18

Operator Precedence:

When a complex expression has multiple operators, the operator precedence (or order of execution of operators) determines the sequence in which the operations are performed.

The order of execution can significantly affect the resulting value.

The operators have the precedence levels as shown in the table given in the slide.

An operator on higher levels is evaluated before an operator on lower level.

3.2: SELECT statement Clauses

The DISTINCT clause

- The SQL DISTINCT clause is used to eliminate duplicate rows.
- For example: Displays student codes from student_marks tables. the student codes are displayed without duplication

```
SELECT DISTINCT student_code  
FROM student_marks;
```



Copyright © Capgemini 2015. All Rights Reserved. 19

The DISTINCT clause:

In the examples discussed so far, some of the values have been repeated. However, by default, all values are retrieved. If you wish to remove duplicate values, then use the query as shown in the slide above.

Retrieval of Constant values by using Dual Table

A “dual” is a table, which is created by Oracle along with the data dictionary. It consists of exactly one column, whose name is dummy, and one record. The value of that record is X.

```
SQL>desc dual;
      Name          Null?    Type
      DUMMY          VARCHAR2(1)
Sql>Select * from dual;
D
-
X
```

The owner of dual is SYS. However, “dual” can be accessed by every user.

As “dual” contains exactly one row (unless someone has fiddled with it), it is guaranteed to return exactly one row in SELECT statements. Therefore, dual

```
SQL>select sysdate from dual;
```

```
SQL>SELECT (319/212)+10 FROM DUAL;
```

For example, you can use it for math:

```
SQL>SELECT employee_seq.NEXTVAL
      FROM DUAL;
```

And, you can use it to increment sequences:

3.2: SELECT statement Clauses

The ORDER BY clause

- The ORDER BY clause presents data in a sorted order.
 - It uses an “ascending order” by default.
 - You can use the DESC keyword to change the default sort order.
 - It can process a maximum of 255 columns.
- In an ascending order, the values will be listed in the following sequence:
 - Numeric values
 - Character values
 - NULL values
- In a descending order, the sequence is reversed.



Copyright © Capgemini 2015. All Rights Reserved. 21

The Order By Clause:

- A query with its various clauses (FROM, WHERE, GROUP BY, HAVING) determines the rows to be selected and the columns. The order of rows is not fixed unless an ORDER BY clause is given.
- An ORDER BY clause is of the form:

ORDER BY < Sort list> ASC/DESC

- The columns to be used for ordering are specified by using the “column names” or by specifying the “serial number” of the column in the SELECT list.
- The sort is done on the column in “ascending” or “descending” order. By default the ordering of data is “ascending” order.

contd.

3.2: SELECT statement Clauses

Sorting Data

- The output of the SELECT statement can be sorted using ORDER BY clause
 - ASC : Ascending order, default
 - DESC : Descending order
- Display student details from student_master table sorted on student_code in descending order.

```
SELECT Student_Code,Student_Name,Dept_Code,  
       Student_dob  
FROM Student_Master  
ORDER BY Student_Code DESC ;
```



Copyright © Capgemini 2015. All Rights Reserved. 22

3.2: SELECT statement Clauses

Sorting Data

- Sorting data on multiple columns

```
SELECT Student_Code,Student_Name, Dept_Code,Student_dob  
      FROM Student_Master  
      ORDER BY Student_Code,Dept_Code;
```



Copyright © Capgemini 2015. All Rights Reserved. 23

The query on the slide sorts the data on both the columns in ascending order which is default. But you could also sort the data in different order for the columns.

For Example in the query given below the data is sorted in ascending order on student_code and dept_code is sorted in descending order

```
SELECT  
      Student_Code,Student_Name,Dept_Code,Student  
      _dob  
      FROM Student_Master  
      ORDER BY Student_Code,Dept_Code DESC;
```

3.3: Tips and Tricks in SELECT Statements

Quick Guidelines

- It is necessary to always include a WHERE clause in your SELECT statement to narrow the number of rows returned.
- If you do not use a WHERE clause, then Oracle will perform a table scan of your table, and return all the rows.
- By returning data you do not need, you cause the SQL engine to perform I/O it does not need to perform, thus wasting SQL Engine resources.



Copyright © Capgemini 2015. All Rights Reserved. 24

Tips and Tricks in SELECT Statements:

It is necessary to always include a WHERE clause in your SELECT statement to narrow the number of rows returned.

In some case you may want to return all rows. Then not using a WHERE clause is appropriate in this case.

However, if you don't need all the rows to be returned, use a WHERE clause to limit the number of rows returned.

Another negative aspect of a table scan is that it will tend to flush out data pages from the cache with useless data. This reduces ability of the Oracle to reuse useful data in the cache, which increases disk I/O and decreases performance.

Quick Guidelines

- In addition, the above scenario increases network traffic, which can also lead to reduced performance.
- And if the table is very large, a table scan will lock the table during the time-consuming scan, preventing other users from accessing it, and will hurt concurrency.
- In your queries, do not return column data that is not required.
 - For example:
 - You should not use SELECT * to return all the columns from a table if all the data from each column is not required.
 - In addition, using SELECT * prevents the use of covered indexes, further potentially decreasing the query performance.



Quick Guidelines

- Carefully evaluate whether the SELECT query requires the DISTINCT clause or not.
 - The DISTINCT clause should only be used in SELECT statements.
 - This is mandatory if you know that "duplicate" returned rows are a possibility, and that having duplicate rows in the result set would cause problems with your application.
 - The DISTINCT clause creates a lot of extra work for SQL Server.
 - The extra load reduces the "physical resources" that other SQL statements have at their disposal.
 - Hence, use the DISTINCT clause only if it is necessary.



Copyright © Capgemini 2015. All Rights Reserved. 26

Tips and Tricks in SELECT Statements (contd.):

Some developers, as a habit, add the DISTINCT clause to each of their SELECT statements, even when it is not required.

This is a bad habit that should be stopped.

Quick Guidelines

- In a WHERE clause, the various “operators” that are used, directly affect the query performance.
- Given below are the key operators used in the WHERE clause, ordered by their performance. The operators at the top produce faster results, than those listed at the bottom.
 - =
 - >, >=, <, <=
 - LIKE
 - <>
- Use “=” as much as possible, and “<>” as least as possible.



Copyright © Capgemini 2015. All Rights Reserved. 27

Tips and Tricks in SELECT Statements (contd.):

Use simple operands

Some operators tend to produce speedy results than other operators. Of course, you may not have choice of using an operator in your WHERE clauses, but sometimes you do have a choice.

Using simpler operands, and exact numbers, provides the best overall performance.

If a WHERE clause includes multiple expressions, there is generally no performance benefit gained by ordering the various expressions in any particular order.

This is because the Query Optimizer does this for you, saving you the effort. There are a few exceptions to this, which are discussed further in the lesson.

contd.

Tips and Tricks in SELECT Statements (contd.):**Don't include code that does not do anything**

- This may sound obvious. However, this scenario is seen in some off-the-shelf applications.

➤ For example, you may see code which is given below:

```
SELECT column_name FROM table_name WHERE  
1 = 0
```

- When this query is run, no rows will be returned. It is just wasting SQL Server resources.
- By default, some developers routinely include code, which is similar to the one given above, in their WHERE clauses when they make string comparisons.

➤ For example:

```
SELECT column_name FROM table_name  
WHERE LOWER(column_name) = 'name'
```

Any use of text functions in a WHERE clause decreases performance.

- If your database has been configured to be case-sensitive, then using text functions in the WHERE clause does decrease performance. However, in such a case, use the technique described below, along with appropriate indexes on the column in question:

```
SELECT column_name FROM table_name  
WHERE column_name = 'NAME' or column_name = 'name'
```

- This code will run much faster than the first example.

Quick Guidelines

- If you use LIKE in your WHERE clause, try to use one or more leading character in the clause, if at all possible.
 - **For example:** Use LIKE 'm%' not LIKE '%m'
- Certain operators in the WHERE clause prevents the query optimizer from using an Index to perform a search.
 - **For example:** "IS NULL", "<>", "!=", "|>", "|<|", "NOT", "NOT EXISTS", "NOT IN", "NOT LIKE", and "LIKE '%500'"



Copyright © Capgemini 2015. All Rights Reserved. 29

Tips and Tricks in SELECT Statements (contd.):

If you use a leading character in your LIKE clause, then the Query Optimizer has the ability to potentially use an Index to perform the query. Thus speeding performance and reducing the load on SQL engine.

However, if the leading character in a LIKE clause is a "wildcard", then the Query Optimizer will not be able to use an Index. Here a table scan must be run, thus reducing performance and taking more time.

The more leading characters you use in the LIKE clause, it is more likely that the Query Optimizer will find and use a suitable Index.

Quick Guidelines

- Suppose you have a choice of using the IN or the BETWEEN clauses. In such a case use the BETWEEN clause, as it is much more efficient.
- For example: The first code is much less efficient than the second code given below.



```
SELECT customer_number, customer_name  
FROM customer  
WHERE customer_number in (1000, 1001, 1002, 1003, 1004)
```

```
SELECT customer_number, customer_name  
FROM customer  
WHERE customer_number BETWEEN 1000 and 1004
```



Copyright © Capgemini 2015. All Rights Reserved. 30

Tips and Tricks in SELECT Statements (contd.):

Assuming there is a useful Index on customer_number, the Query Optimizer can locate a range of numbers much faster by using BETWEEN clause.

This is much faster than it can find a series of numbers by using the IN clause (which is really just another form of the OR clause).

Using Efficient Non-index WHERE clause sequencing:

Oracle evaluates un-indexed equations, linked by the AND verb in a bottom-up fashion. This means that the first clause (last in the AND list) is evaluated, and if it is found TRUE, then the second clause is tested.

Always try to position the most expensive clause first in the WHERE clause sequencing.

Oracle evaluates un-indexed equations, linked by the OR verb in a top-down fashion. This means that the first clause (first in the OR list) is evaluated, and if it is found FALSE, then the second clause is tested.

Always try to position the most expensive OR clause last in the WHERE clause sequencing.

Quick Guidelines

- Do not use ORDER BY in your SELECT statements unless you really need to use it.
- Whenever SQL engine has to perform a sorting operation, additional resources have to be used to perform this task.



Copyright © Capgemini 2015. All Rights Reserved. 31

Tips and Tricks in SELECT Statements (contd.):

Don't use ORDER BY in your SELECT statements unless you really need to:
The ORDER BY clause adds a lot of extra overhead.

For example: Sometimes it may be more efficient to sort the data at the client than at the server. In other cases, the client does not even need sorted data to achieve its goal. The key here is to remember that you should not automatically sort data, unless you know it is necessary.
Whenever SQL Server has to perform a sorting operation, additional resources have to be used to perform this task. Sorting often occurs when any of the following Transact-SQL statements are executed:

ORDER BY
GROUP BY
SELECT DISTINCT
UNION
CREATE INDEX (generally not as critical as happens much less often)

In many cases, these commands cannot be avoided. On the other hand, there are few ways in which sorting overhead can be reduced, like:

Keep the number of rows to be sorted to a minimum. Do this by only returning those rows that absolutely need to be sorted.

Keep the number of columns to be sorted to the minimum. In other words, do not sort more columns than required.

Keep the width (physical size) of the columns to be sorted to a minimum.

Sort column with number datatypes instead of character datatypes.

Summary

- In this lesson, you have learnt:
 - What is SELECT statement?
 - Usage of the following:
 - The WHERE clause
 - The DISTINCT clause
 - The Comparison, Arithmetic, and Logical operators
 - The AND or OR clause
 - The NOT clause
 - The ORDER BY clause



Review – Questions

- Question 1: The ___ table consists of exactly one column, whose name is “dummy”.
- Question 2: The LIKE operator comes under the ___ category.
 - Option 1: mathematical
 - Option 2: comparison
 - Option 3: logical
- Question 3: The ___ specifies the order in which the operators should be evaluated.



Review – Questions

- Question 4: The NOT NULL operator finds rows that do not satisfy a condition.
 - True / False

- Question 5: More than one column can also be used in the ORDER BY clause.
 - True / False



DBMS SQL

Lesson 04: Aggregate
(GROUP) Functions

Lesson Objectives

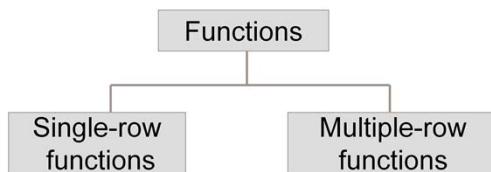
- To understand the following topics:
- Introduction to Functions
- Aggregate (Group) functions:
 - GROUP BY clause
 - HAVING clause



4.1: Introduction to Functions

Types of SQL Functions

- Single row functions :
 - Operate on single rows only and return one result per row
- Multiple row functions:
 - Manipulates groups of rows to give one result per group of rows. Also called as group functions



Copyright © Capgemini 2015. All Rights Reserved 3

Functions:

Functions can be used to manipulate data values in a variety of ways.

Functions help in making the basic query block more powerful.

Functions may be used to perform calculations on data, alter data formats for display, convert data types, etc.

Functions are similar to operators in that they manipulate data items and return a result.

Functions differ from operators in the format in which they appear with their arguments. This format allows them to operate on zero, one, two, or more arguments:

function(argument, argument, ...)

There are two types of functions: SQL functions and User-defined functions.

SQL functions: SQL functions are built into most DBMS and are available for use in various appropriate SQL statements. SQL functions are further categorized as:

Single-Row functions: Single-row functions return a single result row for every row of a queried Table or View.

For example: ABS, ROUND etc.

Aggregate functions: Aggregate functions return a single row based on groups of rows, rather than on single rows. For example: MAX, MIN, COUNT, etc.

User-defined functions: You can write user-defined functions in PL/SQL or Java to provide functionality that is not available in SQL or SQL functions. User-defined functions can appear in a SQL statement wherever SQL functions can appear, that is, wherever an expression can occur.

For example: User-defined functions can be used in the following:

The select list of a SELECT statement

The condition of a WHERE clause

CONNECT BY, START WITH, ORDER BY, and GROUP BY clauses

The VALUES clause of an INSERT statement

The SET clause of an UPDATE statement

4.1: Introduction to Functions

The Group Functions

- The Group functions are built-in SQL functions that operate on “groups of rows”, and return one value for the entire group.
- The results are also based on groups of rows.
- For Example, Group function called “SUM” will help you find the total marks, even if the database stores only individual subject marks.



Copyright © Capgemini 2015. All Rights Reserved

5

Aggregate (Group) Functions:

SQL provides a set of “built-in” functions for producing a single value for an entire group. These functions are called as “Set functions” or “Aggregate (Group) functions”.

These functions can work on a “normal result table” or a “grouped result table”.

If the result is not grouped, then the aggregate will be taken for the whole result table.

4.1: Introduction to Functions

Syntax : GROUP BY & HAVING clause

▪ Syntax

```
SELECT      [column, ] aggregate function(column), .....
FROM        table
[WHERE       condition]
[GROUP BY column]
[HAVING    condition]
[ORDER BY  column] ;
```



Copyright © Capgemini 2015. All Rights Reserved 6

4.1: Introduction to Functions

Listing of Group Functions

- Given below is a list of Group functions supported by SQL:

Function	Value returned
SUM (expr)	Sum value of expr, ignoring NULL values.
AVG (expr)	Average value of expr, ignoring NULL values.
COUNT (expr)	Number of rows where expr evaluates to something other than NULL. COUNT(*) counts all selected rows, including duplicates and rows with NULLs.
MIN (expr)	Minimum value of expr.
MAX (expr)	Maximum value of expr.



Copyright © Capgemini 2015. All Rights Reserved

7

Aggregate (Group) Functions supported by SQL:

All the above functions operate on a number of rows (for example, an entire table), and are therefore known as “Group (or Aggregate) functions”.

A Group function can be used on a subset of the rows in a table by using the WHERE clause.

The Aggregate functions ignore NULL values in the column.

To include NULL values, NVL function can be used with Aggregate functions.

Note :

Count(*) : Returns the number of rows in the table, including duplicates and those with NULLs.

Count(<Expression>) : Returns the number of rows where expression is NOT NULL.

Aggregate (Group) Functions supported by SQL (contd.):
SUM(COL_NAME | EXPRESSION)

- SUM returns the total of values present in a particular “column” or a “number of columns” that are linked together in the expression. All the columns, which form the argument to SUM, must be numeric only.
- To find the sum of one subject for all students following query is used:

```
SELECT SUM(subject1) FROM student_marks;
```

- To find the yearly compensation of staff from department 20, the following query is used:

```
SELECT SUM(12*staff_sal) FROM staff_master  
WHERE dept_code = 20;
```

AVG(COL_NAME | EXPRESSION)

- AVG is similar to SUM. AVG returns the average of a NUMBER of values. The restrictions, which apply on SUM also, apply on AVG.
- To find the average salary the staff, the following query is used:

```
SELECT AVG(sal) FROM staff_master;
```

```
SELECT AVG(12*staff_sal) FROM staff_master  
WHERE dept_code = 20;
```

- To find the average yearly compensation of staff from department 20, the following query is used:

Aggregate (Group) Functions supported by SQL (contd.):**COUNT(*)**

- COUNT returns the number of rows.
- To find the total number of staff members, the following query is used:

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

➤ It is possible to restrict the rows for the operation of COUNT.

- To find the total number of staff members in department 10, the following query is used:

```
SELECT COUNT(*) FROM staff_master  
WHERE dept_code = 10 ;
```

- To find the total number of staff members hired after '01-JAN-02', the following query is used:

```
SELECT COUNT(*) FROM staff_master  
WHERE hiredate >'01-JAN-02' ;
```

➤ When an aggregate function is used in a SELECT statement, column names cannot be used in SELECT unless GROUP BY clause is used.

MIN(COL_NAME | EXPRESSION)

- MIN returns the lowest of the values from the column. MIN accepts columns, which are NON-NUMERIC too.
- To find the minimum salary paid to a staff member, the following query is used:

```
SELECT MIN(staff_sal) FROM staff_master;
```

- To list the staff member who alphabetically heads the list, the following query is used:

```
SELECT MIN(staff_name) FROM staff_master;
```

MAX(COL_NAME | EXPRESSION)

- MAX is the reverse of MIN. MAX returns the maximum value from among the list of values.
- To find the maximum marks for a student in subject2, the following query is used:

```
SELECT MAX(subject2) FROM student_marks
```

4.1: Introduction to Functions

Examples of using Group Functions

- Example 1: Display the total number of records from student_marks.

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

- Example 2: Display average marks from each subject.

```
SELECT AVG(Student_sub1), AVG(Student_sub2), AVG(Student_sub3) FROM  
Student_Marks;
```



Copyright © Capgemini 2015. All Rights Reserved 10

Note:

The first query returns the value, which counts the number of rows fetched from the student_marks table. All the rows in the table are treated as one group. Group Functions operate on sets of rows to give one result per group. Can be used on the whole table or certain set of rows

4.2 : Using the GROUP BY & HAVING clause

The GROUP BY clause

- GROUP BY clause is used along with the Group functions to retrieve data that is grouped according to one or more columns.
- For example: Displays the average staff salary based on every department. The values are grouped based on dept_code

```
SELECT Dept_Code, AVG(Staff_sal)  
      FROM Staff_Master  
     GROUP BY Dept_Code;
```



Copyright © Capgemini 2015. All Rights Reserved 11

Usage of GROUP BY and HAVING clauses:

All the SELECT statements we have used until now have acted on data as if the data is in a “single group”. But the rows of data in some of the tables can be thought of as being part of “different groups”.

For example: The staff_master table contains staff information allocated to various departments identified by dept_code. If we wish to find the minimum salary of each group of employees in respective department, then none of the clauses that we have seen until now are of any use.

The GROUP BY clause is used to group the result table derived from earlier FROM and WHERE clauses. Further, a HAVING clause is used to apply search condition on these groups.

When a GROUP BY clause is used, each row of the resulting table will represent a group having same values in the column(s) used for grouping.

Subsequently, the HAVING clause acts on the resulting grouped table to remove the row that does not satisfy the criteria in the HAVING search condition

The GROUP BY Clause:

If you have to fetch columns other than group functions or if you want your results to be grouped on certain criteria use "GROUP BY" clause

- The GROUP BY clause is of the form:

```
GROUP BY <column list>
```

- The columns specified must be selected in the query. If a table is grouped, the SELECT list should have columns and expressions, which are single-valued for a group. These can be:
 - columns on which grouping is done
 - constants
 - aggregate functions on the other columns on which no grouping is done
- 1. The GROUP BY clause should contain all the columns in the SELECT list, except those used along with the Group functions. Only the column names which have been used in GROUP BY clause and aggregate columns can be used in SELECT clause
- 2. When an Aggregate (Group) function is used in a SELECT statement, the column names cannot be used in SELECT, unless GROUP BY clause is used.
- 3. Grouping can be done on multiple columns, as well.

4.2 : Using the GROUP BY & HAVING clause

The HAVING clause

- HAVING clause is used to filter data based on the Group functions.
- HAVING clause is similar to WHERE condition. However, it is used with Group functions.
- Group functions cannot be used in WHERE clause. However, they can be used in HAVING clause.



Copyright © Capgemini 2015. All Rights Reserved 13

The HAVING Clause:

A HAVING clause is of the form:

HAVING <search condition>

The HAVING search condition applies to “each group”. It can be:

formed using various predicates like between, in, like, null, comparison, etc

combined with Boolean operators like AND, OR, NOT

Since the search condition is for a “grouped table”. The predicates should be:

on a column by which grouping is done.

on a set function (Aggregate function) on other columns.

The aggregate functions can be used in HAVING clause. However, they cannot be used in the WHERE clause.

When WHERE, GROUP BY, and HAVING clauses are used together in a SELECT statement:

The WHERE clause is processed first in order.

Subsequently, 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.

4.2 : Using the GROUP BY & HAVING clause

Examples – GROUP BY and HAVING clause

- For example: Display all department numbers having more than five employees.

```
SELECT Department_Code, Count(*)  
      FROM Staff_Master  
      GROUP BY Department_Code  
      HAVING Count(*) > 5;
```



Copyright © Capgemini 2015. All Rights Reserved 14

The HAVING clause (contd.):

To find out Average, Maximum, Minimum salary of departments, where average salary is greater than 2000.

```
SELECT  
dept_code, AVG(staff_sal), MIN(staff_sal),  
MAX(staff_sal)  
FROM staff_master  
GROUP BY dept_code HAVING  
AVG(staff_sal) > 2000;
```

To find out average salary of all staff members who belong to department 10 or 20 and their average salary is greater than 10000

```
SELECT  
design_code, dept_code, avg(staff_sal)  
      FROM staff_master where dept_code  
      in(10,20)  
      GROUP BY design_code, dept_code  
      HAVING avg(staff_sal) > 10000;
```

4.3: Tips and Tricks on using Group Functions GROUP BY & HAVING clause

Quick Guidelines

- All group functions except COUNT(*) ignores NULL values.
- To substitute a value for NULL values use NVL functions.
- DISTINCT clause makes the function consider only non duplicate values.
- The AVG and SUM are used with numerical data.
- The MIN and MAX functions used with any data type.



Copyright © Capgemini 2015. All Rights Reserved 15

NVL function is covered in the next lesson

Quick Guidelines

- All individual columns included in the SELECT clause other than group functions must be specified in the GROUP BY clause.
- Any column other than selected column can also be placed in GROUP BY clause.
- By default rows are sorted by ascending order of the column included in the GROUP BY list.
- WHERE clause specifies the rows to be considered for grouping.



NVL function is covered in the next lesson

Quick Guidelines

- Suppose your SELECT statement contains a HAVING clause. Then write your query such that the WHERE clause does most of the work (removing undesired rows) instead of the HAVING clause doing the work of removing undesired rows.
- Use the GROUP BY clause only with an Aggregate function, and not otherwise.
- Since in other cases, you can accomplish the same end result by using the DISTINCT option instead, and it is faster.



Copyright © Capgemini 2015. All Rights Reserved 17

Tips and Tricks:

By appropriately using the WHERE clause, you can eliminate unnecessary rows before they reach the GROUP BY and HAVING clause. Thus saving some unnecessary work, and boosting performance.

For example: In a SELECT statement with WHERE, GROUP BY, and HAVING clauses, the query executes in the following sequence.

First, the WHERE clause is used to select the appropriate rows that need to be grouped.

Next, the GROUP BY clause divides the rows into sets of grouped rows, and then aggregates their values.

And last, the HAVING clause then eliminates undesired aggregated groups.

If the WHERE clause is used to eliminate as many of the undesired rows as possible, then the GROUP BY and the HAVING clauses will have to do less work. Thus boosting the overall performance of the query.

contd.

Tips and Tricks (contd.):

The GROUP BY clause can be used with or without an Aggregate function. However, if you want optimum performance, do not use the GROUP BY clause without an Aggregate function. This is because you can accomplish the same end result by using the DISTINCT option instead, and it is faster.

For example: You can write your query two different ways:

```
SELECT OrderID  
FROM [Order Details]  
WHERE UnitPrice > 10  
GROUP BY OrderID
```

or

```
SELECT DISTINCT OrderID  
FROM [Order Details]  
WHERE UnitPrice > 10
```

Both of the above queries produce the same results, but the second one will use less resources and perform faster.

Summary

- In this lesson, you have learnt about:
 - Aggregate (Group functions)
 - GROUP BY clause
 - HAVING clause



Review – Questions

- Question 1: Identify the various group functions from the list given below:
 - Option 1: maximum
 - Option 2: sum
 - Option 3: count
 - Option 4: minimum
- Question 2: The AVG function ignores NULL values in the column.
 - True / False
- Question 3: Count(*) returns the number of rows in the table, including duplicates and those with NULLs.
 - True / False



DBMS SQL

Lesson 05 : SQL (Single-row)
Functions

Lesson Objectives

- To understand the following topics:
 - SQL (single-row) functions
 - Number functions
 - Character functions
 - Date functions
 - Conversion functions
 - Miscellaneous Single-row functions



5.1: Types of Single Row Functions

Single Row Functions

- Single-row functions return a single result row for every row of a queried Table or View.
- Single-row functions can appear in SELECT lists, WHERE clauses, START WITH and CONNECT BY clauses, and HAVING clauses.
- Different categories of single valued functions are:
 - Numerical functions
 - Character functions
 - Date and Time functions
 - Conversion functions
 - Miscellaneous Single-row functions



Copyright © Capgemini 2015. All Rights Reserved. 3

SQL Functions:

So far, we have seen “Aggregate functions”, which operate against a “collection of values”, however return a “single value”.

Now we shall see “scalar functions” which operate against a “single value”, and return a “single value” based on the input value.

The functions can be broadly classified into:

Numerical functions- Accept numeric input and return numeric values

For example: ABS, CEIL, TRUNC, ROUND, POWER, etc.

Character functions - Accept character input and can return both characters and number values

For example: CONCAT, LPAD, RPAD, TRIM, SUBSTR, etc.

Date and Time functions- Operate on values of the DATE data type

For example: SYSDATE, ADD_MONTHS, LAST_DAY, etc.

Conversion functions- Convert a value from one data type to another

For example: CAST, ASCIIISTR, ROWIDTOCHAR, etc.

Miscellaneous Single-row functions

For example: BFILENAME, DECODE, NVL, NULLIF, USERENV, etc.

5.1: Types of Single Row Functions

Single Row Functions - Characteristics

- Manipulate data items
- Accept arguments and return one value
- Act on each row returned
- Return one result per row
- May modify the data type
- Can be nested
- Accept arguments which can be a column or an expression
function_name [(arg1, arg2, ...)]
- Can be used in SELECT, WHERE, and ORDER BY clauses



Copyright © Capgemini 2016. All Rights Reserved. 4

Let us now have a look at each of these function categories

5.1: Types of Single Row Functions

Number Functions

- Number functions accept “numeric data” as argument, and returns “numeric values”.

CEIL (arg)	Returns the smallest integer greater than or equal to “arg”.
FLOOR (arg)	Returns the largest integer less than or equal to “arg”.
ROUND (arg,n)	Returns “arg” rounded to “n” decimal places. If “n” is omitted, then “arg” is rounded as an integer.
POWER (arg, n)	Returns the argument “arg” raised to the n^{th} power.
SQRT (arg)	Returns the square root of “arg”.
SIGN (arg)	Returns -1, 0, or +1 according to “arg” which is negative, zero, or positive respectively.
ABS (arg)	Returns the absolute value of “arg”.
MOD (arg1, arg2)	Returns the remainder obtained by dividing “arg1” by “arg2”.



Copyright © Capgemini 2016. All Rights Reserved. 5

Numeric Functions:

TRUNC(n,m)

The trunc function returns a number truncated to a certain number of decimal places. The syntax for the trunc function is:

trunc(number, [decimal_places])

where:

number = the number to truncate.

decimal_places = the number of decimal places to truncate to. This value must be an integer. If this parameter is omitted, the trunc function will truncate the number to 0 decimal places.

For example:

trunc(125.815) would return 125

trunc(125.815, 1) would return 125.8

trunc(125.815, -1) would return 120

trunc(125.815, -2) would return 100

ROUND(n,m)

The round function returns a number rounded to a certain number of decimal places. The syntax for the round function is:

round(number, [decimal_places])

Numeric Functions (contd.):

where:

number = the number to round.

decimal_places = the number of decimal places rounded to.

This value must be an integer. If this parameter is omitted, the round function will round the number to 0 decimal places.

For example:

round(125.315) would return 125

round(125.315, 0) would return 125

round(125.315, 1) would return 125.3

round(125.315, 2) would return 125.32

round(-125.315, 2) would return -125.32

CEIL(n)

It returns smallest integer greater than or equal to n.

```
SELECT CEIL(15.7) "Ceiling" FROM DUAL;
```

Ceiling
16

FLOOR(n)

•It returns largest integer equal to or less than n.

```
SELECT FLOOR(15.7) "Floor" FROM DUAL;
```

Floor
15

ABS(n)

•It returns the absolute value of n.

```
SELECT ABS(20) "Absolute" FROM DUAL;
```

Absolute
20

POWER function

•It returns m raised to nth power .It is of the form: Power(m,n)

```
SELECT POWER (3,3) "Raised" FROM DUAL;
```

Raised
27

5.1: Types of Single Row Functions

Number Functions - Examples

Example 1:

```
SELECT ABS(-15) "Absolute" FROM dual;
```

Absolute

15

Example 2:

```
SELECT POWER(3,2) "Raised" FROM dual;
```

Raised

9



Copyright © Capgemini 2015. All Rights Reserved. 7

Examples of Number Functions:

DUAL table, which is shown in the slide, is a table owned by SYS.

SYS owns the “data dictionary”, and DUAL is part of the data dictionary.

DUAL is a small work-table, which consists of only one row and one column, and contains the value “x” in that column. Besides arithmetic calculations, it also supports “date retrieval” and it’s “formatting”.

Often a simple calculation needs to be done. A SELECT must have a table name in its FROM clause, else it fails.

To facilitate such calculations via a SELECT, the DUAL dummy table is provided.

The structure of the DUAL table can be viewed by using the SQL statement:

```
DESC DUAL;
```

Number Functions - Examples

Example 3: ROUND(n,m): Returns n rounded to m places

Number

17.2

```
SELECT ROUND(17.175,1) "Number" FROM dual;
```

Example 4: TRUNC(n,m): Returns n rounded to m places

```
SELECT TRUNC(15.81,1) "Number" FROM dual;
```

Number

15.8



Copyright © Capgemini 2015. All Rights Reserved.

Examples of Number (numeric) Functions (contd.):

Round(n,m):

```
SELECT ROUND(17.175,-1) "Number" FROM dual;
```

O/P : 20

TRUNC(n,m):

```
SELECT TRUNC(15.81,-1) "Number" FROM dual;
```

O/P : 10

5.1: Types of Single Row Functions

Character Functions

Character functions accept “character data” as argument, and returns “character” or “number” values.

LOWER (arg)	Converts alphabetic character values to lowercase.
UPPER (arg)	Converts alphabetic character values to uppercase.
INITCAP (arg)	Capitalizes first letter of each word in the argument string.
CONCAT (arg1, arg2)	Concatenates the character strings “arg1” and “arg2”.
SUBSTR (arg, pos, n)	Extracts a substring from “arg”, “n” characters long, and starting at position “pos”.
LTRIM (arg)	Removes any leading blanks in the string “arg”.
RTRIM (arg)	Removes any trailing blanks in the string “arg”.
LENGTH (arg)	Returns the number of characters in the string “arg”.
REPLACE (arg, str1, str2)	Returns the string “arg” with all occurrences of the string “str1” replaced by “str2”.
LPAD (arg, n, ch)	Pads the string “arg” on the left with the character “ch”, to a total width of “n” characters.
RPAD (arg, n, ch)	Pads the string “arg” on the right with the character “ch”, to a total width of “n” characters.



Copyright © Capgemini 2015. All Rights Reserved. 9

Character Functions:

Example1:

```
SELECT Upper('Hello'), Lower('WORLD') FROM Dual;
```

Note: Functions can be nested to any depth. Evaluation starts with the “inner most functions”, and proceeds outwards.

For example: LENGTH(LTRIM(RTRIM(name)))

5.1: Types of Single Row Functions

Character Functions - Examples

Example 1:

```
SELECT CONCAT('Hello ','World')      "Concat" FROM Dual;
```

Concat
Hello World

Example 2:

```
SELECT SUBSTR('HelloWorld',1,5) "SubString" FROM Dual;
```

SubSt
Hello



Copyright © Capgemini 2015. All Rights Reserved. 10

Examples of Character Functions:

REPLACE function returns char with every occurrence of search_string replaced with replacement_string.

If replacement_string is omitted or NULL, then all occurrences of search_string are removed.

If search_string is NULL, then char is returned.

```
SELECT REPLACE('JACK and JUE','J','BL')
"Changes" FROM DUAL;
```

Changes

BLACK and BLUE

CONCAT function returns "char1" concatenated with "char2". Both "char1" and "char2" can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. The string returned is in the same character set as char1. It's datatype depends on the datatypes of the arguments.

In concatenations of two different datatypes, the Oracle Database returns the datatype that results in a "lossless conversion". Therefore:

If one of the arguments is a LOB, then the returned value is a LOB.

If one of the arguments is a national datatype, then the returned value is a national datatype.

String Functions:

- The CONCAT function is equivalent to the “concatenation operator (||)”. The function is useful when there are spaces in the values to be concatenated. The “concatenation operator” does not permit spaces.

UPPER(string)

- This function converts all characters in string to uppercase.

```
SELECT UPPER(staff_name),staff_code
FROM staff_master;
```

LOWER(string)

- This function converts all characters in string to lowercase.

```
SELECT LOWER(student_name) FROM
student_master;
```

INITCAP(string):

- This function converts the first character of each word in string to uppercase and the rest of the characters to lowercase.

```
SELECT INITCAP(staff_name) FROM
staff_master;
```

LPAD(string1,n,string2)

- This function adds string2 before string1 as many times as required to make the string1 length equal to “n” chars.
- To right align the names of staff_members:

```
SELECT LPad(staff_name,30) FROM staff_master;
```

LTRIM(string,CHAR set)

- This function removes chars from beginning of string as long as the character matches one of the chars in the CHAR set.

```
SELECT
student_name,LTRIM(student_name,'MALICE')
FROM student_master;
```

RPAD(string1,n,string2)

- This function is similar to LPAD. It adds chars to the right end.

RTRIM(string,CHAR set)

- This function is similar to LTRIM. It removes chars from the right end.

SUBSTR(CHAR,m,n)

- This function returns the string from the mth character of the string to the nth character

String Functions (contd.):

- **LENGTH function**

It returns the length of char in characters
It is of the form:

Length (string)

Select length ('candide') " length in characters"
From dual;

INSTR(string, pattern[, start [,occurrence]])

- It returns the location of a character IN a STRING.

SELECT INSTR ('CORPORATEFLOOR','R',3,2)
"Instring" FROM DUAL

Instring
6

5.1: Types of Single Row Functions

Date Functions

- Date Functions operate on Date & Time datatype values

Add_Months(date1,int1)	Returns a DATE, int1 times added, int1 can be a negative integer
Month_Between(date1,date2)	Returns number of months between two dates
Last_Day(date1)	Returns the date of the last day of the month that contains the date
Next_Day(date1,char)	Returns the date of the first weekday specified as char that is later than the given date
Current_Date()	Returns the current date in the session time zone. The value is in Gregorian Calendar type
Current_Timestamp	Returns the current date and time in the session time zone. The value returned is ofTimeStamp with TimeZone.
Extract(datetime)	Extracts and returns the value of a specified datetime field
Round(date,[fmt])	Returns date rounded to the unit specified . The format will be according to format model fmt
Trunc(date,[fmt])	Returns date truncated to the unit specified . The format will be according to format model fmt

- Sysdate function returns the current date and time



Copyright © Capgemini 2015. All Rights Reserved. 13

DATE Functions

Datetime functions operate on date (DATE), timestamp, and interval values.

Some of the datetime functions were designed for the Oracle DATE datatype (ADD_MONTHS, CURRENT_DATE, LAST_DAY, and NEXT_DAY).

If you provide a timestamp value as their argument, Oracle Database internally converts the input type to a DATE value and returns a DATE value.

The exceptions are:

the MONTHS_BETWEEN function, which returns a number, and

the ROUND and TRUNC functions, which do not accept timestamp or interval values at all.

The remaining datetime functions are designed to accept any of the three types of data (date, timestamp, and interval), and to return a value of one of these types.

5.1: Types of Single Row Functions

Date Functions- Examples

- Example 1: To display today's date:

```
SELECT sysdate FROM dual;
```

- Example 2: To add months to a date:

```
SELECT ADD_MONTHS(sysdate,10) FROM dual ;
```

- Example 3: To find difference in two dates

```
SELECT MONTHS_BETWEEN(sysdate,'01-sep-95')  
FROM dual ;
```



Copyright © Capgemini 2015. All Rights Reserved 14

Date Functions:

ADD_MONTHS(DATE1,int1) returns the date as addition of “date” and “integer” months. The first argument can be a datetime value or any value that can be implicitly converted to DATE. The second argument can be an integer or any value that can be implicitly converted to an integer. The return type is always DATE.

```
SELECT  
book_code,ADD_MONTHS(book_issue_date,1)  
FROM book_transaction;
```

MONTHS_BETWEEN (DATE1,DATE2) This function returns number of months between the two DATEs. The result is positive if date1 is later than date2 and result is negative if date2 is later than date2.

```
SELECT staff_code,  
MONTHS_BETWEEN(TRUNC(sysdate),hiredate)  
FROM staff_master;
```

Date Functions- Examples

- Example 3: To find out last day of a particular month.

```
SELECT last_day(sysdate) FROM dual ;
```

- Example 4: To find the date of the specified day

```
SELECT next_day(sysdate,'Sunday') FROM dual ;
```

- Example 5: To display date and time according to current time zone set for the database

```
SELECT sessiontimezone,current_date,current_timestamp FROM  
dual ;
```



Copyright © Capgemini 2016. All Rights Reserved. 15

Date Functions (contd.):

LASTDAY(date1) returns the date of the last day of the month that contains the date. The return type is always DATE.

To display the date of the last day in the month the books were issued

```
SELECT book_issue_date,  
LAST_DAY(book_issue_date) FROM  
book_transaction;
```

NEXT_DAY(date1, char) Returns the date of the first weekday specified as char that is later the given date

To display the date on coming Friday in the week that books were issued

```
SELECT book_issue_date,  
NEXT_DAY(book_issue_date,'Friday')  
FROM book_transaction
```

CURRENT_DATE & CURRENT_TIMESTAMP return current date and time respectively base on the timezone set for the database.

The query given on the slide above shows the date and time based on the offset time base on GMT which is according to the timezone set for the database

Date Functions- Examples

- Examples of Extract function

- To extract year from sysdate

```
SELECT EXTRACT (year from sysdate) FROM dual ;
```

- To extract month from date specified

```
SELECT EXTRACT(month FROM DATE '2011-04-01')  
FROM dual;
```

- To extract month of issue for all books

```
SELECT extract(month from book_issue_date)  
FROM book_transaction;
```



Copyright © Capgemini 2015. All Rights Reserved. 10

Date Functions (contd.):

EXTRACT(datetime) extracts the value of a specified datetime field. This function is useful for manipulating datetime field values. For example you can extract only year, month or day from a given date value.

To display year of birth for all students

```
SELECT EXTRACT(year from student_dob) from  
student_master
```

TRUNC (DATE1)

This function truncates the time part from the DATE. This is required when we do DATE calculations.

```
SELECT staff_name, TRUNC(sysdate) -  
TRUNC(HIREDATE) FROM staff_master;
```

Arithmetic with Dates

- Use '+' operator to Add and '-' operator Subtract number of days to/from a date for a resultant date value

```
SELECT Student_code , (Book_actual_return_date -  
Book_expected_return_date) AS Payable_Days  
FROM Book_Transaction  
WHERE Book_Code = 1000;
```



Copyright © Capgemini 2016. All Rights Reserved. 17

Given below are formats of Date Functions

Format	Meaning
YYYY	Four digit year
YY	Last two digits of the year
MM	Month (01-12 where JAN = 01 ...)
MON TH	Name of the month stored as a length of nine characters.
MON	Name of the month in three letter format.
DD	Day of the month (01-31).
D	Day of the week.
DAY	Name of the day stored as a length of nine characters.
DY	Name of the day in three letter format.
FM	This prefix can be added to suppress blank padding.
HH	Hour of the day.
HH24	Hour in the 24 hour format.
MI	Minutes of the hour.
SS	Seconds of the minute.
TH	The suffix used with the day.

5.1: Types of Single Row Functions

Conversion Functions

- Conversion functions facilitate the conversion of values from one datatype to another.

TO_CHAR (arg,fmt)	Converts a number or date "arg" to a specific character format.
TO_DATE (arg,fmt)	Converts a date value stored as string to date datatype
TO_NUMBER (arg)	Converts a number stored as a character to number datatype.
TO_TIMESTAMP(arg,fmt)	Converts character type to a value of timestamp datatype



Copyright © Capgemini 2015. All Rights Reserved. 18

Conversion Functions

Conversion functions convert values of one datatype to another. Usually the conversion function accepts two arguments wherein first is the input type and second is the output type.

Oracle takes care of implicit datatype conversion. Explicit datatype conversions are done using the conversion functions.

Although Oracle does provide implicit type conversion to ensure reliability of SQL statements you should use conversion functions

Conversion Functions - Examples

Example 1: To display system date in format as 29 November, 1999.

```
SELECT to_char(sysdate,'DD month, YYYY) FROM dual ;
```

Example 2: To display system date in the format as 29th November, 1999.

```
SELECT to_char(sysdate,'DDth month,YYYY') FROM dual ;
```



Copyright © Capgemini 2016. All Rights Reserved. 19

TO_CHAR(DATE1,format)

This function converts the DATE given to the format specified.

```
SELECT TO_CHAR(TRUNC(sysdate), 'ddth  
fmMonth yy') 'DATE'  
FROM dual;
```

DATE

01st January 95

```
SELECT TO_CHAR(TRUNC(sysdate),'fmMonth')  
"DATE" FROM dual;
```

```
SELECT to_char ( sysdate , 'Q') FROM dual ;
```

DATE

July

Example: To display the quarter which has the specified date.

Conversion Functions - Examples

Example 3: To display employees whose hiredate is September 08,1981.

```
SELECT staff_code, hiredate FROM staff_master  
WHERE  
hiredate = TO_DATE ('September 08,1981','Month DD, YYYY');
```

Example 4: To display the value in timestamp format

```
SELECT TO_TIMESTAMP(sysdate,'DD-MM-YY') from dual;
```



Copyright © Capgemini 2016. All Rights Reserved. 20

5.1: Types of Single Row Functions

Miscellaneous Functions

- Some functions do not fall under any specific category and hence listed as miscellaneous functions

NVL (arg1,arg2)	Replaces and returns a null value with specified actual value
NVL2(arg1,arg2,arg3)	If arg1 is not null then it returns arg2. If arg1 is null then arg3 is returned
NULLIF(arg1,arg2)	Compares both the arguments, returns null if both are equal or first argument if both are unequal
COALESCE(arg1,arg2 ...argn)	Returns the first non null value in the given list
CASE	Both these functions are for conditional processing, with this IF-Then-Else logic can be applied in SQL statements
DECODE	



Copyright © Capgemini 2016. All Rights Reserved. 21

Miscellaneous Functions

These functions are sometimes also called as General Functions.
These functions work with any datatype values

Miscellaneous Functions - Examples

Example 1: To display the return date of books and if not returned it should display today's date

```
SELECT book_code,  
NVL(book_actual_return_date,sysdate)  
FROM book_transaction;
```

Example 2: To examine expected return date of book, and if null return today's date else return the actual return date

```
SELECT book_code,  
NVL2(book_expected_return_date,book_actual_return_date, sysdate)  
FROM book_transaction;
```



Copyright © Capgemini 2016. All Rights Reserved. 22

Miscellaneous Single-row Functions:

NVL()

Many times there are records holding NULL values in a table. When an output of such a table is displayed, it is difficult to understand the reason of NULL or BLANK values shown in the output.

The only way to overcome this problem is to replace NULL values with some other meaningful value while computing the records. This can be done using the NVL function.

NVL()

The NVL2 function can be thought of as an extension to NVL function. But this function examines the first value. If the value is not null then returns the second value and if null then returns the third value.

Examples for both functions are shown on the slide

Miscellaneous Functions - Examples

Example 3: To check if the actual return date of the book is same as the expected return date of the book

```
SELECT book_code,  
NULLIF(book_expected_return_date, book_actual_return_date)  
FROM book_transaction;
```

Example 4: To track whether the expected/actual return date of the book is populated if any of the values is null it will display sysdate

```
SELECT book_code, COALESCE(book_expected_return_date,  
book_actual_return_date, sysdate)  
FROM book_transaction;
```



Copyright © Capgemini 2015. All Rights Reserved. 23

NULLIF()

This function compares the arguments provided. If they are equal, the function returns null and if not then it returns the first argument.

You cannot specify the literal NULL as the first argument.

COALESE()

The Coalesce function returns the first non null value is the given argument list. The benefit of using this function versus NVL() us that it can take multiple alternate values.

Examples for both the functions are shown on the slide

5.1: Types of Single Row Functions

The Case Function

▪ Case() function

Conditional evaluation by doing work of an IF-THEN-ELSE statement
Syntax

```
CASE expr when compare_expr1 then return_expr1
           [when compare_exprn then return_exprn
            ELSE else_expr]
            END
```

Example

```
CASE expr when compare_expr1 then return_expr1
           [when compare_exprn then return_exprn
            ELSE else_expr]
            END
```



Copyright © Capgemini 2015. All Rights Reserved. 24

CASE()

In a simple case expression, Oracle searches for the first WHEN...THEN pair for which expr is equal to comparison expr and returns return_expr.

The expressions used should be of same datatype.

CASE is capable of more logical comparisons like <> etc..

Also CASE can work with predicates and subqueries

5.1: Types of Single Row Functions

The Decode Function

Decode () function:

Similar to CASE, Conditional evaluation by doing work of an IF-THEN-ELSE statement
Syntax

Example:

```
DECODE (<exp or coln>, <val1>,<o/p1>,<val2>,<o/p2>,
....., <default o/p>)
```

```
SELECT staff_code, staff_name, dept_code,
       DECODE (deptno,10,'Ten',20,'Twenty','Others')
  FROM staff_master WHERE design_code = 102;
```



Copyright © Capgemini 2016. All Rights Reserved. 25

DECODE () function

This function decodes the expression after comparing it to each search value. If the expression is the same as search, result is returned. If the default value is omitted, a null value is returned where a search value does not match any of the result values.

As compared to CASE, DECODE can do an equality check only. The expressions in DECODE can work only on scalar values.

DECODE can work as a function inside SQL only but CASE can be more efficient substitute in PL/SQL blocks.

5.2: Tips and Tricks

Quick Guidelines

- If possible, try avoiding the SUBSTRING function in the WHERE clauses.
- Depending on how it is constructed, using the SUBSTRING function can force a table scan instead of allowing the Optimizer to use an Index (assuming there is one).
- Instead, use the LIKE condition, for better performance.
 - For example: Use the second query instead of using the first query.

```
WHERE SUBSTRING(column_name,1,1) = 'b'
```

```
WHERE column_name LIKE 'b%'
```



Copyright © Capgemini 2016. All Rights Reserved. 20

Tips and Tricks:

If possible, try avoiding the SUBSTRING function in your WHERE clauses.

Depending on how it is constructed, using the SUBSTRING function can force a table scan instead of allowing the Optimizer to use an Index (assuming there is one).

If the substring you are searching for does not include the first character of the column you are searching for, then a table scan is performed.

In case of conversion functions, If value to be converted is not in right format, then Oracle will throw an error

Summary

- In this lesson, you have learnt:
 - SQL (single-row) functions
 - Character functions
 - Number functions
 - Date functions
 - Conversion functions
 - Miscellaneous Single-row functions



Review – Questions

- Question 1: Single row functions can be broadly classified as ____.
 - Option 1: character functions
 - Option 2: numeric functions
 - Option 3: Date functions
 - Option 4: all the above
- Question 2: The function which returns the value after capitalizing the first character is ____.



Review – Questions

- Question 3: The output of the following function will be ____.
 - select to_char(17000,'\$99,999.00') 'Amount' from dual;
- Question 4: The decode function can convert NULL values to required type.
 - True / False
- Question 5: The function which returns the last date of the month is ____.
 - Option 1: LAST_DATE
 - Option 2: LAST_DAY
 - Option 3: MONTH_LAST_DATE
 - Option 4: MONTH_LAST_DAY



DBMS SQL

Lesson 6: Joins and Subqueries

Lesson Objectives

- To understand the following topics:
 - Join
 - Oracle Proprietary Joins
 - SQL: 1999 Compliant Joins
 - Sub-queries
 - Co-related sub-query
 - Exists / Non-Exists Operators
 - CONNECT BY and START WITH clauses



6.1: Joins

What are Joins?

- If we require data from more than one table in the database, then a join is used.
- Tables are joined on columns, which have the same “data type” and “data width” in the tables.
- The JOIN operator specifies how to relate tables in the query.
 - When you join two tables a Cartesian product is formed, by default.
- Oracle supports
 - Oracle Proprietary
 - SQL: 1999 Compliant Joins



Copyright © Capgemini 2015. All Rights Reserved 3

Joins:

JOINS make it possible to select data from more than one table by means of a single statement.

The joining of tables is done in SQL by specifying the tables to be joined in the FROM clause of the SELECT statement.

When you join two tables a Cartesian product is formed.

The conditions for selecting rows from the product are determined by the predicates in the WHERE clause.

All the subsequent WHERE, GROUP BY, HAVING, ORDER BY clauses work on this product.

If the same table is used more than once in a FROM clause then “aliases” are used to remove conflicts and ambiguities. They are also called as “co-relation names” or “range variables”.

Joins (contd.):

Assume two tables:

TABLE F1

<u>COL1</u>	<u>COL2</u>
A	1
B	2
C	3

Table F2

<u>COL1</u>	<u>COL2</u>
X	100
Y	200

The statement `SELECT * FROM F1,F2;` results in:

<u>COL1</u>	<u>COL2</u>	<u>COL1</u>	<u>COL2</u>
A	1	X	100
B	2	X	100
C	3	X	100
A	1	Y	200
B	2	Y	200
C	3	Y	200

6 rows selected

The statement `SELECT * FROM F1 First_T, F1 Second_T;`
results in:

<u>COL1</u>	<u>COL2</u>	<u>COL1</u>	<u>COL2</u>
A	1	A	1
B	2	A	1
C	3	B	2
A	1	B	2
B	2	C	3
C	3	C	3

9 rows selected

6.1: Joins

Types of Joins

- Given below is a list of JOINS supported by Oracle:

Oracle Proprietary Joins	SQL: 1999 Compliant Joins
Cartesian Product	Cross Joins
Equijoin	Inner Joins (Natural Joins)
Outer-join	Left, Right, Full outer joins
Non-equi-join	Join on
Self-join	Join on



Copyright © Capgemini 2015. All Rights Reserved.

5

Note:

Oracle9i onwards offers JOIN syntax that is SQL: 1999 compliant.

Prior to the 9i release, the JOIN syntax was different from the ANSI standards.

The new SQL: 1999 compliant JOIN syntax does not offer any performance benefits over the Oracle proprietary JOIN syntax that existed in prior releases.

As we go ahead we will see both variations of joins supported by Oracle.

6.2: Oracle Proprietary Joins

Cartesian Joins

- A Cartesian product is a product of all the rows of all the tables in the query.
- A Cartesian product is formed when the join condition is omitted or it is invalid
- To avoid having Cartesian product always include a valid join condition

Example

```
SELECT Student_Name, Dept_Name  
FROM Student_Master, Department_Master;
```



Copyright © Capgemini 2015. All Rights Reserved



Cartesian Product

Whenever a join condition is completely omitted or is invalid a Cartesian product results. It displays all combinations of rows. A Cartesian product tends generates a large number of rows. Unless there is some specific need to combine all rows avoid a Cartesian product by including a valid join condition in the query

The example shown on the slide joins all rows of Student_Master and Department Master resulting in a Cartesian Join

Guidelines for Joining Tables

- The JOIN condition is written in the WHERE clause
- The column names which appear in more than one table should be prefixed with the table name
- To improve performance of the query, table name prefix can be include for the other selected columns too



Copyright © Capgemini 2015. All Rights Reserved 7

Before we get on to Joins let us understand some basic guidelines to write Join Queries

6.2: Oracle Proprietary Joins

EquiJoin

- In an Equijoin, the WHERE statement compares two columns from two tables with the equivalence operator “=”.
 - This JOIN returns all rows from both tables, where there is a match.
- Syntax :

```
SELECT <col1>, <col2>,...  
FROM <table1>,<table2>  
Where <table1>.<col1>=<table2>.<col2>  
[AND <condition>] [ORDER BY <col1>, <col2>,...]
```



Copyright © Capgemini 2015. All Rights Reserved

Equi Join

Equi Join which is sometimes also referred to as Inner Join or simple join is done by writing a join condition using the “=” operator

Typically the tables are joint to get meaningful data.

The join is based on the equality of column values in the two tables and therefore is called an Equijoin.

To join together “n” tables, you need a minimum of “n-1” JOIN conditions.

For example: To join three tables, a minimum of two joins is required.

In the syntax given in the slide:

Column1 in Table1 is usually the Primary key of that table.

Column2 in Table2 is a Foreign key in that table.

Column1 and Column2 must have the same data type, and for certain data types, they should have same size, as well.

6.2: Oracle Proprietary Joins

EquiJoin - Example

Example 1: To display student code and name along with the department name to which they belong

```
SELECT Student_Code,Student_name,Dept_name  
FROM Student_Master ,Department_Master  
WHERE Student_Master.Dept_code =  
      Department_Master.Dept_code;
```

Example 2: To display student and staff name along with the department name to which they belong

```
SELECT student_name,staff_name, dept_name  
FROM student_master, department_master,staff_master  
WHERE student_master.dept_code=department_master.dept_code  
and staff_master.dept_code=department_master.dept_code;
```



Copyright © Capgemini 2015. All Rights Reserved

Equi Join

Frequently, these type of JOIN involves PRIMARY and FOREIGN key complements.

You can also use table aliases to qualify column names in the SELECT and Join Condition

6.2: Oracle Proprietary Joins

Non-EquiJoin

- A non-equi join is based on condition other than an equality operator
- Example: To display details of staff_members who receive salary in the range defined as per grade

```
SELECT s.staff_name,s.staff_sal,sl.grade  
FROM staff_master s,salgrade sl  
WHERE staff_sal BETWEEN sl.losal and sl.hisal
```



Copyright © Capgemini 2015. All Rights Reserved 10

Non-Equijoin

A Non-equijoin is a JOIN condition containing something other than an equality operator.

The example on the slide shows a non-equijoin operation

Assume that we have a Salgrade table which is used to determine the range of salary for all staff member. The structure of the table is as follows:

Name	Type
GRADE	NUMBER
LOSL	NUMBER
HISAL	NUMBER

So to display all the staff members who receive salary between the ranges specified in the salgrade table we will use a non-equijoin

6.2: Oracle Proprietary Joins

Outer Join

- If a row does not satisfy a JOIN condition, then the row will not appear in the query result.
- The missing row(s) can be returned by using OUTER JOIN operator in the JOIN condition.
- The operator is PLUS sign enclosed in parentheses (+), and is placed on the side of the join(table), which is deficient in information.

```
WHERE table1 <OUTER JOIN INDICATOR> = table 2
```



Copyright © Capgemini 2015. All Rights Reserved T1

Outer Join

If a row does not satisfy the join condition, the row will not appear in the query result. In this situation outer join can be used

Outer Joins are similar to Inner Joins. However, they give a bit more flexibility when selecting data from related tables. This type of join can be used in situations where it is desired to select “all rows from the table on the left or right”, regardless whether they match the join condition

Outer Join is an exclusive “union” of sets (whereas normal joins are intersection). OUTER JOINs can be simulated using UNIONs.

In a JOIN of two tables an Outer Join may be for the first table or the second table. If the Outer Join is taken on, say the DEPARTMENT_MASTER table, then each row of this table will be selected at least once whether or not a JOIN condition is satisfied.

An Outer Join does not require each record in the two joint tables to have a matching record in the other table. The joint table retains each record — even if there is no other matching record.

Outer Join

Syntax

- Table1.column = table2.column (+) means OUTER join is taken on table1.
- The (+) sign must be kept on the side of the join that is deficient in information
- Depending on the position of the outer join (+), it can be denoted as Left Outer or Right outer Join



Copyright © Capgemini 2015. All Rights Reserved 13

Outer Join (contd.):

The plus(+) operator can appear only on one side of the expression. It returns those rows from one table that have no direct match in the other table.

One restriction on outer join is that you cannot use IN operator or the OR operator to create a complex condition

Outer Join - Example

- To display Department details which have staff members and also display department details who do not have any staff members

```
SELECT staff.staff_code,staff.Dept_Code,dept.Dept_name  
FROM Staff_master staff, Department_Master dept  
WHERE staff.Dept_Code(+) = dept.Dept_Code
```



Copyright © Capgemini 2015. All Rights Reserved 13

6.2: Oracle Proprietary Joins

Self Join

- In Self Join, two rows from the “same table” combine to form a “resultant row”.
- It is possible to join a table to itself, as if they were two separate tables, by using aliases for table names.
- This allows joining of rows in the same table.

Example: To display staff member information along with their

- manager information

```
SELECT staff.staff_code, staff.staff_name,  
       mgr.staff_code, mgr.staff_name  
  FROM staff_master staff, staff_master mgr  
 WHERE staff.mgr_code = mgr.staff_code;
```



Copyright © Capgemini 2015. All Rights Reserved 14

Self Join:

Sometimes is required to join the table to itself. To join a table to itself, “two copies” of the same table have to be opened in the memory.

Since the table names are the same, the second table will overwrite the first table. In effect, this will result in only one table being in memory.

This is because a table name is translated into a specific memory location.

Hence in the FROM clause, the table name needs to be mentioned twice with an “alias”

These two table aliases will cause two identical tables to be opened in different memory locations.

This will result in two identical tables to be physically present in the computer memory.

6.3: SQL:1999 Compliant Joins

SQL: 1999 Compliant Joins - Syntax

- Syntax:

```
SELECT table1.column, table2.column  
FROM table1  
[CROSS JOIN table2] |  
[NATURAL JOIN table2] |  
[JOIN table2 USING (column_name)] |  
[JOIN table2 ON (table1.column_name =  
    table2.column_name)] |  
[LEFT|RIGHT|FULL OUTER JOIN table2  
ON (table1.column_name = table2.column_name)];
```



Copyright © Capgemini 2015. All Rights Reserved 15

SQL:1999 Compliant Joins

The SQL Compliant joins can obtain the similar results that we have discussed in the previous slides. They differ only in syntax.

The SQL compliant joins are supported from Oracle 9i version onwards

6.3: SQL:1999 Compliant Joins

Cross Join

- The Cross Join and Cartesian product are same which produces the cross-product of the tables

Example: Cross Join on Student_Master and Department_Master

```
SELECT student_name, dept_name  
FROM student_master  
CROSS JOIN department_master;
```



Copyright © Capgemini 2015. All Rights Reserved 16

Cross Join

The example on the slide create a cross product of the two tables.
The query result is same as the following query:

```
SELECT student_name,dept_name  
FROM Student_Master, Department_Master;
```

6.3: SQL:1999 Compliant Joins

Natural Join

- The Natural Join is based on the all columns that have same name and datatype in the tables include in the query
- All the rows that have equal values in the matched columns are fetched

Example: To display student details along with their department details

```
SELECT Student_Code,Student_name,Dept_Code,  
       Dept_name  
  FROM Student_Master  
NATURAL JOIN Department_Master
```



Copyright © Capgemini 2015. All Rights Reserved 17

Natural Join

Prior to Oracle 9i, without explicitly specifying the columns of the corresponding tables a join was not possible. With the Natural Join clause, the join can happen automatically based on column names that match in name and datatype.

Oracle returns an error if the datatypes of the columns are different though they have the same name.

The Natural Join is same as EquiJoin.

6.3: SQL:1999 Compliant Joins

USING clause

- The USING clause can be replace the NATURAL JOIN if the columns have same names but data types do not match.
- The table name or aliases should not be used in the referenced columns
- This clause should be used to match only one column when there are more than one column matches



Copyright © Capgemini 2015. All Rights Reserved 18

USING clause

The NATURAL JOIN returns an error if the datatypes of matching columns are different. In that case the USING clause can be used to specify only those columns on which the join has to done.

The NATURAL JOIN and USING clause are mutually exclusive.

The column used in USING clause cannot be used in WHERE clause

USING clause - Example

Example 1: To display student details along with their department details. The department code does not match in datatype, hence the join is performed with the USING clause

```
SELECT student_code, student_name, dept_code, dept_name  
FROM student_master  
JOIN department_master  
USING (dept_code, dept_code);
```

6.3: SQL:1999 Compliant Joins

ON clause

- Explicit join condition can be specified by using ON clause
- Other search conditions can be specified in addition to join condition

Example: To display student along with department details from Computer Science department

```
SELECT student.student_code, student.student_name,  
student.dept_code, dept.dept_name  
FROM student_master student  
JOIN department_master dept  
ON (student.dept_Code = dept.dept_Code)  
AND dept.dept_Name ='Computer Science' ;
```



Copyright © Capgemini 2015. All Rights Reserved 20

ON clause

With the ON clause you can specify join conditions separate from any other search conditions.

The query is readable and easy to understand

6.3: SQL:1999 Compliant Joins

LEFT, RIGHT & FULL Outer Join

- A join between two tables that return rows that match the join condition and also unmatched rows from left table is LEFT OUTER JOIN
- A join between two tables that return rows that match the join condition and unmatched rows from the right table is RIGHT OUTER JOIN
- A join between two tables that return rows that match the join condition and returns unmatched rows of both left and right table is a full outer join



Copyright © Capgemini 2015. All Rights Reserved 21

LEFT, RIGHT and FULL OUTER JOIN

The SQL compliant outer join is similar to Oracle proprietary outer join except for the fact that it also has support to perform FULL OUTER JOIN

LEFT, RIGHT & FULL Outer Join - Example

Example 1: Display student & department details and also those departments who do not have students

```
SELECT s.student_code, s.dept_code, d.dept_name  
FROM student_master s  
RIGHT OUTER JOIN department_master d  
ON (s.dept_code = d.dept_code);
```

Example 2 Display student & department details, also those students who are not assigned to any department

```
SELECT s.student_code, s.dept_code, d.dept_name  
FROM student_master s  
LEFT OUTER JOIN department_master d  
ON (s.dept_code = d.dept_code);
```



Copyright © Capgemini 2015. All Rights Reserved 23

LEFT, RIGHT & FULL Outer Join - Example

Example 3: Display student & department details. Also those departments who do have students and students who are not assigned to any department

```
SELECT s.student_code,s.dept_code,d.dept_name  
FROM student_master s  
FULL OUTER JOIN department_master d  
ON (s.dept_code = d.dept_code );
```



Copyright © Capgemini 2015. All Rights Reserved 23

6.4: Subqueries

What is a SubQuery?

- A sub-query is a form of an SQL statement that appears inside another SQL statement.
- It is also called as a “nested query”.
- The statement, which contains the sub-query, is called the “parent statement”.
- The “parent statement” uses the rows returned by the sub-query.



Copyright © Capgemini 2015. All Rights Reserved 24

Sub-queries:

As mentioned earlier, since a basic SQL query returns a relation, it can be used to construct composite queries.

Such a SQL query, which is nested within another higher level query, is called a “sub-query”.

This kind of a nested sub-query is useful in cases, where we need to select rows from tables based on a condition, which depends on the data stored in the table itself.

These can be useful when you need to select rows from a table with a condition that depends on data within the table itself.

6.4: Subqueries

Subquery - Examples

Example 1: To display name of students from “Mechanics” department.

Method 1:

```
SELECT Dept_Code FROM Department_Master  
WHERE Dept_name = 'Mechanics';
```

O/P : 40

```
SELECT student_code,student_name FROM student_master  
WHERE dept_code=40;
```



Copyright © Capgemini 2015. All Rights Reserved 25

Consider the example given on the slide. We want to find details of students from “Mechanics” department. As you can see two queries are used to resolve these problem. Instead of that you can resolve this problem by combining both the queries into one as shown on the next slide.

Subquery - Examples

Example 1 (contd.):

Method 2: Using sub-query

```
SELECT student_code, student_name  
FROM student_master  
WHERE dept_code = (SELECT dept_code  
                    FROM department_master  
                    WHERE dept_name = 'Mechanics');
```



Copyright © Capgemini 2015. All Rights Reserved 26

The problem is resolved using a one query inside another. The inner query is called as a subquery or nested query. The subquery result is used by the outer query. The subqueries can have more levels of nesting. The innermost query will execute first. The subquery execute only once before the outer query

Where to use Subqueries?

- Subqueries can be used for the following purpose :
 - To insert records in a target table.
 - To create tables and insert records in the table created.
 - To update records in the target table.
 - To create views.
 - To provide values for conditions in the clauses, like WHERE, HAVING, IN, etc., which are used with SELECT, UPDATE and DELETE statements.



Copyright © Capgemini 2015. All Rights Reserved 27

- When the WHERE clause needs a set of values which can be only obtained from another query, the Sub-query is used. In the WHERE clause it can become a part of the following predicates

- COMPARISON Predicate
- IN Predicate
- ANY or ALL Predicate
- EXISTS Predicate.

- It can be also used as a part of the condition in the HAVING clause.

6.3: Subqueries

Comparison Operators for Subqueries

- Types of SubQueries
 - Single Row Subquery
 - Multiple Row Subquery.
- Some comparison operators for subqueries:

Operator	Description
IN	Equals to any member of
NOT IN	Not equal to any member of
*ANY	compare value to every value returned by sub-query using operator *
*ALL	compare value to all values returned by sub-query using operator *



Copyright © Capgemini 2015. All Rights Reserved 28

Sub-queries by using Comparison operators:

Sub-queries are divided as “single row” and “multiple row” sub-queries. While single row comparison operators ($=$, $>$, \geq , $<$, \leq , \neq) can only be used in single row sub-queries, multiple row sub-queries can use IN, ANY or ALL.

For example: The assignment operator ($=$) compares a single value to another single value. In case a value needs to be compared to a list of values, then the IN predicate is used. The IN predicate helps reduce the need to use multiple OR conditions.

The NOT IN predicate is the opposite of the IN predicate. This will select all rows where values do not match the values in the list.

The FOR ALL predicate is evaluated as follows:

True if the comparison is true for every value of the list of values.

True if sub-query gives a null set (No values)

False if the comparison is false for one or more of the list of values generated by the sub-query.

The FOR ANY predicate is evaluated as follows

True if the comparison is true for one or more values generated by the sub-query.

False if sub-query gives a null set (No values).

False if the comparison is false for every value of the list of values generated by the sub-query.

Using Comparison Operators - Examples

Example 1: To display all staff details of who earn salary least Salary

```
SELECT staff_name, staff_code, staff_sal  
FROM staff_master  
WHERE staff_sal = (SELECT MIN(staff_sal)  
                   FROM staff_master);
```

Example 2: To display staff details who earn salary greater than average salary earned in dept 10

```
SELECT staff_code, staff_sal FROM staff_master  
WHERE staff_sal > ANY(SELECT AVG(staff_sal)  
                      FROM staff_master WHERE dept_code=10);
```



Copyright © Capgemini 2015. All Rights Reserved 29

For Single Row Subquery the sub-query should result in one value of the same data type as the left-hand side.

Similarly for Multiple Row Subquery the list of values generated by the sub-query should be of same data type as the left-hand side

The slide above shows examples of Subqueries. The first query is an example of Single Row Subquery and the second is an example of Multiple row Subquery

6.4: Co-related Subquery

What is a Co-related Subquery?

- A sub-query becomes “co-related”, when the sub-query references a column from a table in the “parent query”.
- A co-related sub-query is evaluated once for each row processed by the “parent statement”, which can be either SELECT, UPDATE, or DELETE statement.
- A co-related sub-query is used whenever a sub-query must return a “different result” for each “candidate row” considered by the “parent query”.



Copyright © Capgemini 2015. All Rights Reserved 30

Co-related Sub-query:

A “co-related sub-query” is a form of query used in SELECT, UPDATE, or DELETE commands to force the DBMS to evaluate the query once “per row” of the parent query, rather than once for the “entire query”.

A co-related sub-query is used to answer “multi-part questions”, whose answers depend on the values in each row of the parent query.

A co-related sub-query is one way of reading every row in a table, and comparing values in each row against related data.

Co-related Subquery -Examples

- Example 2: To display staff details whose salary is greater than the average salary in their own department:

```
SELECT staff_name, staff_sal , dept_code  
FROM staff_Master s  
WHERE staff_sal > (SELECT AVG(staff_sal)  
FROM staff_Master m  
WHERE s.dept_code = m.dept_code );
```



Copyright © Capgemini 2015. All Rights Reserved 31

6.4: Co-related Subquery

EXISTS/ NOT EXISTS Operator

- The EXISTS / NOT EXISTS operator enables to test whether a value retrieved by the Outer query exists in the result-set of the values retrieved by the Inner query.
- The EXISTS / NOT EXISTS operator is usually used with a co-related sub-query.
 - If the query returns at least one row, the operator returns TRUE.
 - If the value does not exist, it returns FALSE.
- The NOT EXISTS operator enables to test whether a value retrieved by the Outer query is not a part of the result-set of the values retrieved by the Inner query.



Copyright © Capgemini 2015. All Rights Reserved

30

The Exists Predicate is of the form

<Query> WHERE
 EXISTS < sub-query>

Query will be evaluated if EXISTS takes a value TRUE. It takes the Value TRUE if the <sub-query> result table is non null and FALSE if it is Null. It is mostly used with Co-Related Subqueries. EXISTS does not check for a particular value. It checks whether subquery returns rows or not.

EXISTS/ NOT EXISTS Operator - Examples

- Example 1: To display details of employees who have some other employees reporting to them.

```
SELECT staff_code, staff_name FROM staff_master staff  
WHERE EXISTS (SELECT mgr_code FROM staff_master mgr WHERE  
mgr.mgr_code = staff.staff_code);
```

- Example 2: To display details of departments which have employees working in it.

```
SELECT dept_code,dept_name FROM department_master  
WHERE EXISTS ( SELECT dept_code FROM staff_master  
WHERE staff_master.dept_code =  
department_master.dept_code);
```



Copyright © Capgemini 2015. All Rights Reserved 33

CONNECT BY and START WITH Clauses

- The START WITH .. CONNECT BY clause can be used to select data that has a hierarchical relationship
- Usually, they have some sort of parent-child relationship.
- They are used to retrieve rows, which are connected to each other through a tree-like structure.



Copyright © Capgemini 2015. All Rights Reserved 34

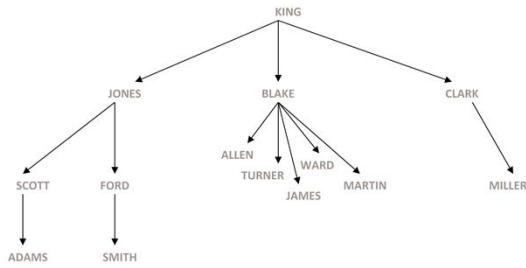
CONNECT BY and START WITH Clauses:

A pseudocolumn behaves like a table column, but is not actually stored in the table. You can select from pseudocolumns, but you cannot insert, update, or delete their values.

For each row returned by a hierarchical query, the LEVEL pseudocolumn returns 1 for a root row, 2 for a child of a root, and so on. A root row is the highest row within an inverted tree. A child row is any nonroot row. A parent row is any row that has children. A leaf row is any row without children

CONNECT BY and START WITH Clauses

- The earliest ancestor in the tree is called the root-node called as a trunk. Extending from the trunk are branches, which have other branches.



CONNECT BY and START WITH Clauses

- The restrictions on SELECT statements performing hierarchical queries are as follows :
 - A SELECT statement that performs a hierarchical query cannot perform a JOIN.
 - If an ORDER BY clause is used in a hierarchical query, then Oracle orders rows using the ORDER BY clause rather than in a hierarchical fashion.



Copyright © Capgemini 2015. All Rights Reserved 26

CONNECT BY,START WITH Clauses-Examples

- Example 1: To list “Allen” and his subordinates

```
SELECT staff_name, staff_code, mgr_code  
FROM staff_master  
CONNECT BY PRIOR staff_code = mgr_code  
START WITH staff_name = 'Allen';
```

Note: If START WITH clause is omitted, then the tree structure is generated for each of the rows in the EMP table.



Copyright © Capgemini 2015. All Rights Reserved 37

6.5: Tips and Tricks

Quick Guidelines

- For Using Subqueries
- Should be enclosed in parenthesis
- They should be placed on the right side of the comparison condition
- Cannot use ORDER By clause in subquery unless performing top-n analysis
- Use operator carefully. Single Row operators for Single Row Subquery and Multiple Row operator for Multiple Row Subquery



Quick Guidelines

- Restrict using the NOT IN clause, which offers poor performance because the optimizer has to use a nested table scan to perform this activity.
- Instead try to use one of the following options, all of which offer better performance:
 - Use EXISTS or NOT EXISTS
 - Use IN
 - Perform a LEFT OUTER JOIN and check for a NULL condition



Quick Guidelines

- If you have a choice of using the IN or the EXISTS clauses in your SQL, use the EXISTS clause as it is usually more efficient and performs faster.
 - Consider EXISTS in place of table joins.
 - Consider NOT EXISTS in place of NOT IN.



Summary

- In this lesson, you have learnt:
 - Joins
 - Oracle Proprietary Joins
 - SQL: 1999 Compliant Joins
- Sub-queries
 - Co-related sub-query
 - Exists / Non-Exists Operators
- CONNECT BY and START WITH clauses



Review – Match the Following

- | | |
|------------------|---|
| 1. Equi Join | a. is based on any other operator other than equality |
| 2. Non-equi join | b. Is based on equality operator |
| 3. Outer Join | c. Joins the table to itself |
| 4. Self Join | d. includes a “+” operator with equality operator |



Review – Questions

- Question 1: The SQL compliant join which is same as EquiJoin.
 - Option 1: Cross Join
 - Option 2: Natural Join
 - Option 3: Full Outer Join

- Question 2: A sub-query is also sometimes termed as ____.



Review – Questions

- Question 3: A sub-query can be used for creating and inserting records.
 - True / False

- Question 4: If a sub-query returns multiple values, then the valid operators is/are ____.
 - Option 1: =
 - Option 2: IN
 - Option 3: >
 - Option 4: Any



DBMS SQL

Lesson 07 : Introduction to Data
Modeling E-R Model and
Normalization

Table Of Contents

- Introduction to Data Modeling
- E-R Model
- Normalization



Copyright © Capgemini 2015. All Rights Reserved 2

Definition of a Model

- An integrated collection of concepts for describing data, relationships between data, and constraints on the data used by an organization.
- A representation of 'real world' objects and events, and their associations.
- It attempts to represent the data requirements of the organization that you wish to model
- Modeling is an integral part of the design and development of any system.
- A correct model is essential.



Copyright © Capgemini 2015. All Rights Reserved 3

What is a model?

A model serves two primary purposes:

As a true representation of some aspects of the real world, a model enables clearer communication about those aspects.

A model serves as a blueprint to shape and construct the proposed structures in the real world.

So, what is a data model? A data model is an instrument that is useful in the following ways:

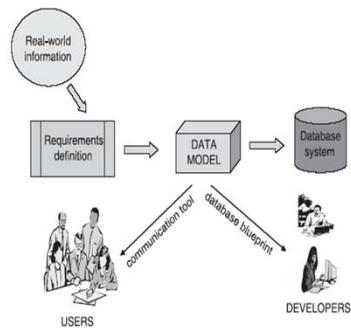
- 1) A model helps the users or stakeholders clearly understand the database system that is being implemented. It helps them understand the system with reference to the information requirements of an organization.
- 2) It enables the database practitioners to implement the database system exactly conforming to the information requirements.

A data model, therefore, serves as a critical tool for communication with the users and it also serves as a blueprint of the database system for the developers.

Without a proper data model, an adequate database system cannot be correctly designed and implemented. A good data model forms an essential prerequisite for any successful database system. Unless the data modelers represent the information requirements of the organization in a proper data model, the database design will be totally ineffective.

What is Data Modeling?

- Data modeling is a technique for exploring the data structures needed to support an organization's information need.
- It would be a conceptual representation or a replica of the data structure required in the database system.
- A data model focuses on which data is required and how the data should be organized.
- At the conceptual level, the data model is independent of any hardware or software constraints.



Copyright © Capgemini 2015. All Rights Reserved 4

What is Data Modeling?

At this level, the data model is generic; it does not vary whether you want to implement an object-relational database, a relational database, a hierarchical database, or a network database.

At the next level down, a data model is a logical model relating to the particular type of database relational, hierarchical, network, and so on. This is because in each of these types, data structures are perceived differently.

If you proceed further down, a data model is a physical model relating to the particular database management system (DBMS) you may use to implement the database.

Why Use Data Modeling?

- Leverage
 - Data model serves as a blueprint for the database system.
- Conciseness
 - Data model functions as an effective communication tool for discussions with the users.
- Data Quality
 - Data model acts as a bridge from real-world information to database storing relevant data content.



Copyright © Capgemini 2015. All Rights Reserved 5

Why Use Data Modeling?

Leverage: The key reason for giving special attention to data organization is the leverage. A small change to a data model may have a major impact on the whole system. Therefore, you can opt for modifying the data model instead of the system. For the most commercial information systems, the programs are far more complex. Also, considerable time is consumed in specifying and constructing them, as compared to the database. However, their contents and structures are heavily influenced by the database design.

Conciseness: A data model is a very powerful tool for establishing requirements and capabilities of information systems. Its valuable because of its conciseness. It implicitly defines a whole set of screens, reports, and processes needed to capture, update, retrieve, and delete the specified data. The data modeling process can tremendously facilitate our understanding of the essence of business requirements.

Data Quality: The data held in a database is usually a valuable business asset built up over a long period. Inaccurate data (poor data quality) reduces the value of the asset and can be expensive or impossible to correct. Frequently, problems with data quality can be traced back to a lack of consistency in (a) defining and interpreting data, and (b) implementing mechanisms to enforce the definitions.

Entity-Relationship Model

- A high-level conceptual data model that is widely used in the design of a database application.
- A top-down approach to database design.
- The ER model represents data in terms of these:
 - Entities (Independent, dependent or Associative)
 - Attributes of entities
 - Relationships between entities
- It is independent of the h/w or s/w used for implementation. Although we can use an ER model as a basis for hierarchical, network and relational databases, it is strongly connected to the relational databases.



Copyright © Capgemini 2015. All Rights Reserved 6

What is Entity-Relationship Model?

Entities are the principal data objects about which information is to be collected. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database. Some specific examples of entities are EMPLOYEES, PROJECTS, INVOICES. An entity is analogous to a table in the relational model.

Attributes

Attributes describe the entity with which they are associated. A particular instance of an attribute is a value. The domain of an attribute is the collection of all possible values an attribute can have. The domain of Name is a character string.

Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys, uniquely identify an instance of an entity. A descriptor describes a non-unique characteristic of an entity instance.

Entities

- People, places, objects, things, events, or concepts (nouns).
- Examples: Employee, salesperson, sale, account, department.
- Represented as a rectangle in ERD.
- Entity type versus entity instance:
 - Entity instance: A single example of an entity.
 - Entity type: A collection of all entity instances of a single type.



SALESPERSON
* Salesperson Number
Salesperson Name
Commission Percentage
Year of Hire



Copyright © Capgemini 2015. All Rights Reserved 7

The basic concept of the ER model is an **entity**, which represents a set of objects in the 'real world' that share the same properties. Each object, which should be uniquely identifiable within the set, is called an **entity occurrence**. An entity has an independent existence and can represent objects with a physical (or 'real') existence or objects with a conceptual (or 'abstract') existence

Attributes

- Attributes describe the entity with which they are associated.
- They are characteristics of an Entity.

Eg: Customer Name is an attribute of the entity Customer.



Copyright © Capgemini 2015. All Rights Reserved 8

Types of Attributes

- Simple Attribute: An attribute composed of a single component with an independent existence. E.g SSN, BirthDate
- Composite Attribute: An attribute composed of multiple components, each with an independent existence. E.g Name attribute of the entity that can be subdivided into FirstName ,LastName attributes



Copyright © Capgemini 2015. All Rights Reserved 9

Attributes

A particular instance of an attribute is a value. For example, "S Ranjan" is one of the values of the attribute Name. The domain of an attribute is the collection of all possible values an attribute can have. The domain of Name is a character string.

Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys, uniquely identify an instance of an entity. A descriptor describes a non-unique characteristic of an entity instance.

Types of Attributes

- Single-Valued Attribute: An attribute that holds a single value for each occurrence. E.g. SSN
- Multi-Valued Attributes: An attribute that holds multiple values for each occurrence. E.g Hobbies
- Derived Attributes: An attribute that represents a value that is derivable from the value of a related attribute or set of attributes, not necessarily in the same entity type.
 - E.g. Age whose value is derived from the Current Date and BirthDate attribute.



Copyright © Capgemini 2015. All Rights Reserved 10

Attributes

Attributes describe the entity with which they are associated. A particular instance of an attribute is a value. For example, "S Ranjan" is one of the values of the attribute Name. The domain of an attribute is the collection of all possible values an attribute can have. The domain of Name is a character string.

Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys, uniquely identify an instance of an entity. A descriptor describes a non-unique characteristic of an entity instance.

Relationship

- A relationship represents an association between two or more entities. An example of a relationship would be as follows:
 - Employees are assigned to projects.
 - Projects have subtasks.
 - Departments manage one or more projects.



Copyright © Capgemini 2015. All Rights Reserved 11

Key Types

Superkey:

- An attribute, or set of attributes, that uniquely identifies each entity occurrence.
- If we add additional attributes to a primary key, the resulting combination would still uniquely identify an instance of the entity set. Such augmented keys are called superkey

Candidate Key:

- A superkey that contains only the minimum number of attributes necessary for unique identification of each entity occurrence. Ex. BranchNo in entity Branch.
- The minimal super key is also referred as candidate key



Copyright © Capgemini 2015. All Rights Reserved 12

Key Types

- Primary Key: The candidate key that is selected to uniquely identify each occurrence of an entity type. E.g: National Insurance Number.
- Alternate keys: The candidate keys that are not selected as the primary key of the entity.
- Composite Key: A candidate key that consists of two or more attributes.



Copyright © Capgemini 2015. All Rights Reserved 13

Primary Keys: The candidate key that is selected to identify each entity occurrence.

They are a strong concept that is usually enforced for every table.

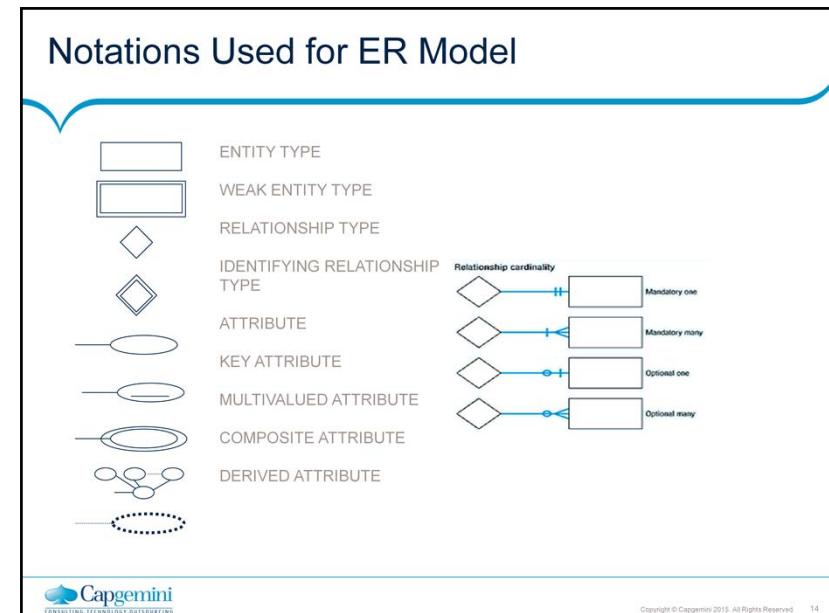
- They can be made up of one or more columns; each has to be mandatory.
- They are declarative as a constraint and can be named. When creating a primary key constraint, Oracle automatically creates a unique index in association with it.
- A foreign key usually refers to the primary key of a table, but may also refer to a unique key.

Tables that do not have a primary key should have a unique key.

Unique Keys: A unique key is a key that for some reason was not selected to be the primary key. The reasons may have been:

- Allowed nulls. Nulls may be allowed in Unique keys columns.
- Updatable. Unique key values may change but still need to remain unique. For example, the home phone number of an employee or the license plate for a car.

There may be more than one unique key for each table.



Notations Used for ER Model:

Entities are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.

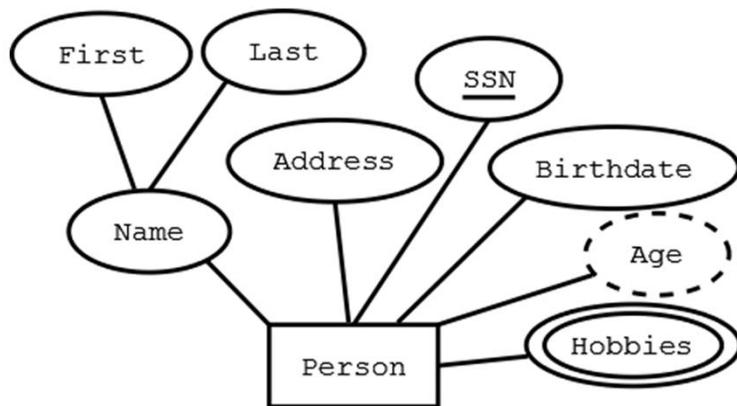
Relationships are represented by a solid line connecting two entities. The name of the relationship is written above the line. Relationship names should be verbs.

Attributes, when included, are listed inside the entity rectangle. Attributes which are identifiers are underlined. Attribute names should be singular nouns.

Cardinality of many is represented by a line ending in a crow's foot. If the crow's foot is omitted, the cardinality is one.

Existence is represented by placing a circle or a perpendicular bar on the line. Mandatory existence is shown by the bar (looks like the number 1) next to the entity that has a mandatory instance. Optional existence is shown by placing a circle next to the entity that is optional.

Attribute Notations



Copyright © Capgemini 2015. All Rights Reserved 15

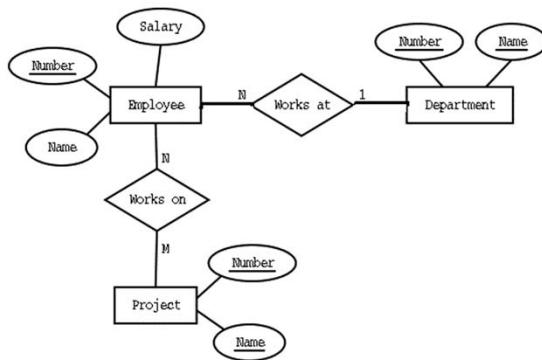
Rectangle -- Entity

Ellipses -- Attribute (underlined attributes are [part of] the primary key)

Double ellipses -- multi-valued attribute

Dashed ellipses-- derived attribute, e.g. age is derivable from birthdate and current date.

The E-R Model - An Example



The Different ER Relationships (cont..)

- Degree of Relationships

- The degree of a relationship is the number of entities associated with the relationship



Copyright © Capgemini 2015. All Rights Reserved 17

Degree of Relationship

The degree of a relationship is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Special cases are the binary and ternary, with the degree 2 and 3 respectively.

Binary relationship is the most common type of association in the real world and is defined between two entities. A recursive binary relationship occurs when an entity is related to itself. An example might be "some employees are married to other employees".

A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are disintegrated into two or more binary relationships.

Degree of a Relationship

- Unary:
 - Recursive relationship between instances of ONE entity type.
 - Example: Salesperson MANAGES Salesperson.
- Binary:
 - Relationship between instances of TWO entity types (most common).
 - Example: Salesperson SELLS product.
- Ternary:
 - Relationship among instances of THREE entity types.
 - Example: Salesperson SELLS product TO customer.



Copyright © Capgemini 2015. All Rights Reserved 18

The entities involved in a particular relationship are referred to as participants. The number of participants in a relationship is called the **degree** and indicates the number of entities involved in a relationship. A relationship of degree one is called **unary**, which is commonly referred to as a *recursive* relationship. A relationship of degree two is called **binary**. The two relationships shown. A relationship of a degree higher than binary is called a **complex relationship**. A relationship of degree three is called **ternary**. A relationship of degree four is called **quaternary**, and a relationship of a higher degree is called **n-ary**. The most popular type of relationship you'll come across is binary, but occasionally you'll come across unary or ternary, and less frequently quaternary.

Types of Binary Relationships

- One-to-one (1:1) binary relationship:
 - One instance of an entity type is related to only one instance of another entity type, and vice versa.
 - Line 3.1.1
 - Example: A salesperson can have only ONE office; an office can be assigned to only ONE salesperson.
- One-to-many (1:N) binary relationship:
 - One instance of an entity type (the “one” side) is related to many instances of another entity type (the “many” side), but an entity instance on the “many” side can be related to only one instance on the “one” side.
 - Example: A salesperson can have MANY sales transaction, but one sales transaction can only be done by ONE salesperson.



Copyright © Capgemini 2015. All Rights Reserved 19

Types of Binary Relationships

- Many-to-many (M:N) binary relationship:
 - Instances of each entity type can be related to many instances of the other entity type.
 - Example: A salesperson can sell MANY products, and a product can be sold by MANY salespersons.



Copyright © Capgemini 2015. All Rights Reserved 20

Normalization

- Normalization is a technique for designing relational database tables.
- Normalization is used:
 - to minimize duplication of information
 - to safeguard the database against certain types of problems
- Thus, Normalization is a process of efficiently organizing data in a database.
- Normalization is done within the framework of five normal forms(numbered from first to fifth).
- Most designs conform to at least the third normal form.
- Normalization rules are designed to prevent anomalies and data inconsistencies.



Copyright © Capgemini 2015. All Rights Reserved 21

Goals of the Normalization

- Goals of the Normalization process are:
 - to eliminate redundant data
 - to ensure that data dependencies make sense
- Thus Normalization:
 - reduces the amount of space a database has consumed
 - ensures that data is logically stored



Copyright © Capgemini 2015. All Rights Reserved 22

Normalization (contd.):

- Higher degrees of normalization typically involve more tables and create the need for a larger number of joins, which can reduce performance.
- Accordingly:
 - More highly normalized tables are typically used in database applications involving many isolated transactions
For example: an Automated teller machine
 - While less normalized tables tend to be used in database applications that need to map complex relationships between data entities and data attributes
For example: a reporting application, or a full-text search application
- Database theory describes a table's degree of normalization in terms of Normal Forms of successively higher degrees of strictness.
For example: A table in Third Normal form (3NF), is consequently in Second Normal form (2NF) as well but the reverse is not necessarily true.
- Although the Normal Forms are often defined informally in terms of the characteristics of tables, rigorous definitions of the Normal Forms are concerned with the characteristics of mathematical constructs known as "Relations". Whenever information is represented relationally, it is meaningful to consider the extent to which the representation is normalized.

Benefits of Normalization

- Benefits of Normalization are given below:
 - Greater overall database organization
 - Reduction of redundant data
 - Data consistency within the database
 - A much more flexible database design
 - A better handle on database security



Copyright © Capgemini 2015. All Rights Reserved 23

Problems with un-normalized database

- An un-normalized database can suffer from various “logical inconsistencies” and “data operation anomalies”.
- Data Operation anomalies can be classified as:
 - Update anomaly
 - Insertion anomaly
 - Deletion anomaly



Copyright © Capgemini 2015. All Rights Reserved 24

The purpose of Normalization is to ensure:

there is no duplication of data

there is no unnecessary data stored within a Database

all the attributes (columns) of a table are completely dependent on the primary key of a table only, and not on any other attribute.

Update anomaly

- The slide shows an Update anomaly.
- Employee 519 is shown as having different addresses on different records.

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry



Copyright © Capgemini 2015. All Rights Reserved 25

Update anomaly:

The same information can be expressed on multiple records.

Therefore updates to the table may result in logical inconsistencies.

For example: Each record in an “Employees’ Skills” table might contain an Employee ID, Employee Address, and Skill. Thus a change of address for a particular employee will potentially need to be applied to multiple records (one for each of his skills). If the update is not carried through successfully — that is, if the employee’s address is updated on some records but not on others — then the table is left in an inconsistent state.

Specifically, the table provides conflicting answers to the question of what is the address of a particular employee. This phenomenon is known as an “Update anomaly”.

The slide shows an Update anomaly. Employee 519 is shown as having different addresses on different records.

Insertion anomaly

- The slide shows an example of an Insertion anomaly.
- Until the new faculty member is assigned to teach at least one course, the details cannot be recorded.

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

424	Dr. Newsome	29-Mar-2007	?
-----	-------------	-------------	---



Copyright © Capgemini 2015. All Rights Reserved 26

Insertion anomaly:

There are circumstances in which certain facts cannot be recorded at all.

For example: Each record in “Faculty and Their Courses” table might contain a Faculty ID, Faculty Name, Faculty Hire Date, and Course Code — thus we can record the details of any faculty member who teaches at least one course. However, we cannot record the details of a newly-hired faculty member who has not yet been assigned to teach any courses. This phenomenon is known as an “Insertion anomaly”.

The slide shows an Insertion anomaly. Until the new faculty member is assigned to teach at least one course, his details cannot be recorded.

Deletion anomaly

- The slide shows an example of a Deletion anomaly.
- All information about Dr. Giddens is lost when he temporarily ceases to be assigned to any course.

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

DELETE



Copyright © Capgemini 2015. All Rights Reserved 27

Deletion Anomaly:

- There are circumstances in which the deletion of data representing certain facts necessitates the deletion of data representing completely different facts. The "Faculty and Their Courses" table described in the previous example suffers from this type of anomaly. If a faculty member temporarily ceases to be assigned to any course, we must delete the last of the records on which that faculty member is displayed. This phenomenon is known as a "Deletion anomaly".
- The slide shows a Deletion anomaly. All information about Dr. Giddens is lost when he temporarily ceases to be assigned to any courses.

Background to Normalization: Functional dependency

- Let us suppose we have two columns A and B, such that, for given value of column A there is a single value of column B associated with it. Then column B is said to be functionally dependent on column A.
- Column B is functionally dependent on column A is equivalent to saying that column A determines(identifies) column B.
 - The same can be notified as $A \rightarrow B$
- Eg: Employee Address has a functional dependency on Employee ID because a particular Employee ID value corresponds to one and only one Employee Address value.
 - $\text{EmpID} \rightarrow \text{EmpAddress}$



Copyright © Capgemini 2015. All Rights Reserved 28

Note that the reverse need not be true. Several employees can live at the same address and therefore one Employee Address value can correspond to more than one Employee ID. Employee ID is therefore not functionally dependent on Employee Address.

An attribute may be functionally dependent either on a single attribute or on a combination of attributes. It is not possible to determine the extent to which a design is normalized without understanding what functional dependencies apply to the attributes within its tables. Understanding this concept, in turn, requires knowledge of the problem domain.

Background to Normalization (contd.):**Three Types of Functional Dependencies**

- **Full Dependency:** In a relation, the attribute(s) B is fully functional dependent on A, if B is functionally dependent on A but not on any proper subset of A.
- **Partial Dependency:** It is a relation, where there is some attribute that can be removed from A and the dependency still holds.
For example: Staff_No, Sname → Branch_No
- **Transitive Dependency:** In a relation, if attribute(s) A→B and B→C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C)
For example: Staff_No → Branch_No, and Branch_No → BAddress
For example: {Employee Address} has a functional dependency on {Employee ID, Skill}, but not a full functional dependency, because it is also dependent on {Employee ID}.
 - **Trivial functional dependency:** A trivial functional dependency is a functional dependency of an attribute on a superset of itself. {Employee ID, Employee Address} → {Employee Address} is trivial, as is {Employee Address} → {Employee Address}.
 - **Transitive dependency:** A transitive dependency is an indirect functional dependency, one in which X → Z only by virtue of X→Y and Y→Z.
 - **Multivalued dependency:** A multi-valued dependency is a constraint according to which the presence of certain rows in a table implies the presence of certain other rows. Refer the Multivalued Dependency article for a rigorous definition.
 - **Join dependency:** A table T is subject to a join dependency if T can always be recreated by joining multiple tables each having a subset of the attributes of T.
 - **Superkey:** A super key is an attribute or set of attributes that uniquely identifies rows within a table. In other words, two distinct rows are always guaranteed to have distinct superkeys. {Employee ID, Employee Address, Skill} will be a superkey for the "Employees' Skills" table; {Employee ID, Skill} will also be a superkey.
 - **Candidate key:** A candidate key is a minimal superkey, that is a superkey for which we can say that no proper subset of it is also a superkey. {Employee Id, Skill} will be a candidate key for the "Employees' Skills" table.
 - **Non-prime attribute:** A non-prime attribute is an attribute that does not occur in any candidate key. Employee Address will be a non-prime attribute in the "Employees' Skills" table.
 - **Primary key:** Most DBMSs require a table to be defined as having a single unique key, rather than a number of possible unique keys. A primary key is a key which the database designer has designated for this purpose.

Normal Forms

- Normal Form is abbreviated as NF.
- Normal Form provides criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies.
- The higher the NF applicable to a table, the less vulnerable it is to inconsistencies and anomalies.



Copyright © Capgemini 2015. All Rights Reserved 30

Normal Forms:

- The Normal Forms (abbrev. NF) of relational database theory provide criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies. The higher the Normal Form applicable to a table, the less vulnerable it is to inconsistencies and anomalies.
- Each table has a “highest normal form” (HNF). By definition, a table always meets the requirements of its HNF and of all normal forms lower than its HNF. Again by definition, a table fails to meet the requirements of any Normal Form higher than its HNF.
- The Normal Forms are applicable to individual tables. To say that an entire database is in normal form n implies that all its tables are in normal form n.
- Newcomers to database design sometimes suppose that normalization proceeds in an iterative fashion, i.e. a 1NF design are first normalized to 2NF, then to 3NF, and so on. This is not an accurate description of how normalization typically works. A sensibly designed table is likely to be in 3NF on the first attempt. Furthermore, if it is 3NF, it is overwhelmingly likely to have an HNF or 5NF. Achieving the “higher” normal forms (above 3NF) does not usually require an extra expenditure of effort on the part of the designer, because 3NF tables usually need no modification to meet the requirements of these higher normal forms.
- Edgar F. Codd originally defined the first three normal forms (1NF, 2NF, and 3NF). These normal forms have been summarized as requiring that all non-key attributes be dependent on “the key, the whole key and nothing but the key”. The fourth and fifth normal forms (4NF and 5NF) deal specifically with the representation of many-to-many and one-to-many relationships among attributes. Sixth normal form (6NF) incorporates considerations relevant to temporal databases.

Un-normalized Relation

- An Unnormalized NF relation is a table that contains one or more Repeating Groups or Non-Atomic values.



Copyright © Capgemini 2015. All Rights Reserved 31

Case Study: Normalization

- Given an EmpProject Table structure
- Note: EmpProject is an un-normalized relation

Proj Code	Proj Type	Proj Desc	Empno	Ename	Grade	Sal scale	Proj Join Date	Alloc Time
001	APP	LNG	46	JONES	A1	5	12/1/1998	24
001	APP	LNG	92	SMITH	A2	4	2/1/1999	24
001	APP	LNG	96	BLACK	B1	9	2/1/1999	18
004	MAI	SHO	72	JACK	A2	4	2/4/1999	6
004	MAI	SHO	92	SMITH	A2	4	5/5/1999	6



Copyright © Capgemini 2015. All Rights Reserved 32

Normalization

- Problems with the above kind of implementation
 - Data Redundancy : Proj. type and description are unnecessarily repeated many times.
 - An employee can't logically exist.
 - If project type changes many records will require updating in an identical manner.
 - If employee Smith gains promotion, many records will require updating.
 - If management decides to change the relationship between the grade and salary scale then many updates will be required.



Copyright © Capgemini 2015. All Rights Reserved 33

Normalization – First Normal Form (1 NF)

- A relation is said to be in “First Normal Form” (1NF) if and only if all its attributes assume only atomic values and there are no repeating groups.
 - 1NF requires that the values in each column of a table are “atomic”.
 - “Atomic” implies that there are no sets of values within a column.
- To Transform un-normalized relation in 1 NF:
 - Eliminate repeating groups.
 - Make a separate table for each set of related attributes, and give each table a primary key.



Copyright © Capgemini 2015. All Rights Reserved 34

First Normal Form (1NF):

- A table is in first normal form (1NF) if and only if it faithfully represents a relation. Given that database tables embody a relation-like form, the defining characteristic of one in First Normal form is that it does not allow duplicate rows or nulls. To put it simply, a table with a unique key (which, by definition, prevents duplicate rows) and without any nullable columns is in 1NF.
- **Note:**
 - The restriction on nullable columns as a 1NF requirement, as espoused by Chris Date, et. al., is controversial. This particular requirement for 1NF is a direct contradiction to Dr. Codd's vision of the relational database, in which he stated that “null values” must be supported in a fully relational DBMS in order to represent “missing information and inapplicable information in a systematic way, independent of data type”.
 - By redefining 1NF to exclude nullable columns in 1NF, no level of normalization can ever be achieved unless all nullable columns are completely eliminated from the entire database. This is in line with Date's and Darwen's vision of the perfect relational database, however it can introduce additional complexities in SQL databases to the point of impracticality.

contd.

Normalization – First Normal Form (1 NF)

Rule:

- A relation is in the first normal form if the intersection of any column and row contains only one value
- Identify any suitable primary key
- Remove repeating groups to simplify relationship that may lead to multiple table design

Table I

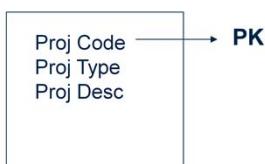
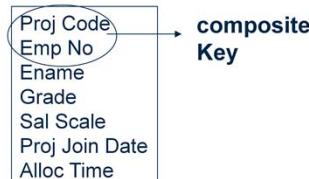


Table II



Copyright © Capgemini 2015. All Rights Reserved 35

Normalization – Second Normal Form (2 NF)

- A relation is in Second Normal Form (2NF) if and only if it is in 1NF and every non-key attribute is fully functionally dependent on the complete primary key of the relation.
- 2NF is based on the concept of Full Functional Dependency.



Copyright © Capgemini 2015. All Rights Reserved 36

Second Normal Form (2NF):

- Where the First Normal Form deals with atomicity of data, the Second Normal Form (or 2NF) deals with relationships between composite key columns and non-key columns. As stated earlier, the normal forms are progressive, so to achieve Second Normal form, your tables must already be in First Normal Form.
- In case of the Second Normal form (or 2NF), any non-key columns must depend on the entire primary key. In the case of a composite primary key, this means that a non-key column cannot depend on only part of the composite key.

Normalization – Second Normal Form (2 NF)

- Transform 1 NF to 2 NF :
 - If a non-key attribute depends on only part of a composite key,
 - remove it to a separate table.
 - For every relation with a single data item making up the primary key, this rule should “always be true”.
 - For those with the composite key, examine every column and ask whether its value depends on the whole of the composite key or just some part of it.
 - Remove those that depend only on part of the key to a new relation with that part as the primary key.



Copyright © Capgemini 2015. All Rights Reserved 37

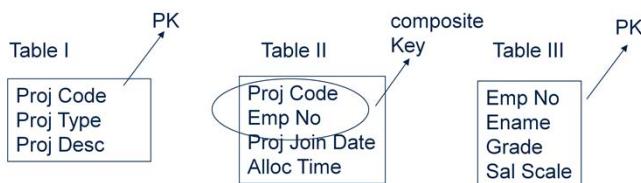
Rules for 2NF:

The Second Normal Form involves the idea of **Functional Dependency**.

- **Functional Dependency:** Attribute B has a functional dependency on attribute A, if for each value of attribute A, there is exactly one value of attribute B.
- **For example:** In an Emp table, Employee Address has a functional dependency on Employee ID because a particular Employee ID value corresponds to one and only one Employee Address value.
- Note that the reverse need not be true.
For example: Several employees can live at the same address, and therefore one Employee Address value can correspond to more than one Employee ID. Employee ID is therefore not functionally dependent on Employee Address.
- An attribute may be functionally dependent either on a single attribute or on a combination of attributes. It is not possible to determine “the extent to which a design is normalized” without understanding the “functional dependencies” that apply to the attributes within its tables.
- Understanding this concept, in turn, requires knowledge of the problem domain.
For example: An Employer may require certain employees to split their time between two locations, such as New York City and London, and therefore want to allow Employees to have more than one Employee Address. In this case, Employee Address will no longer be functionally dependent on Employee ID.

contd.

Normalization – Second Normal Form (2 NF)



Normalization – Third Normal Form (3 NF)

- A relation is in 3NF if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key of the relation.
- 3NF is based on the concept of transitive dependency.



Copyright © Capgemini 2015. All Rights Reserved 39

Third Normal Form (3NF):

- Third Normal Form (3NF) requires that all columns depend directly on the primary key. Tables violate the Third Normal Form when one column depends on another column, which in turn depends on the primary key (a transitive dependency).
- One way to identify transitive dependencies is to look at your table, and see if any columns require updating if another column in the table is updated. If such a column exists, it probably violates 3NF.
- Third Normal Form: Eliminate Data Not Dependent On Key

Normalization – Third Normal Form (3 NF)

- Rules for 3NF are:

- Examine every non-key column and question its relationship with every other non-key column.
- If there is a transitive dependency, then remove both the columns to a new relation.

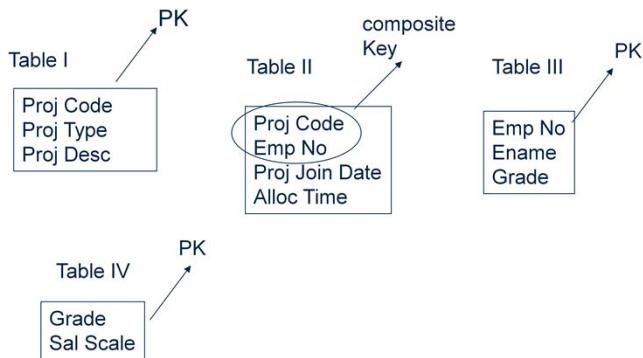


Copyright © Capgemini 2015. All Rights Reserved 40

Criteria for 3NF:

- The table must be in 2NF.
- Every non-prime attribute of the table must be non-transitively dependent on each candidate key. A violation of 3NF will mean that at least one non-prime attribute is only indirectly dependent (transitively dependent) on a candidate key.
- **For example:**
 - Consider a “Departments” table whose attributes are Department ID, Department Name, Manager ID, and Manager Hire Date. And suppose that each manager can manage one or more departments. {Department ID} is a candidate key.
 - Although Manager Hire Date is functionally dependent on the candidate key {Department ID}, this is only because Manager Hire Date depends on Manager ID, which in turn depends on Department ID.
 - This transitive dependency means the table is not in 3NF unless the manager can be hired for more than one department and the date represents the hiring date of that manager only for this department.

Normalization – Third Normal Form (3 NF)



Normalization Summarization

- 1 NF- Ensure all values are atomic and Eliminate Repeating Groups
- 2 NF- Eliminate Partial Dependencies
- 3 NF- Eliminate Transitive Dependencies



CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 42

Drawbacks of Normalization

- Typically, in a normalized database, more joins are required to pull together information from multiple tables.
- Joins require additional I/O to process, and are therefore more expensive from a performance standpoint than single-table lookups.
- Additionally, a normalized database often incurs additional CPU processing. CPU resources are required to perform join logic and to maintain data and referential integrity.



Copyright © Capgemini 2015. All Rights Reserved 43

Summary

- In this lesson, you have learnt that:
 - Data Modeling
 - E-R Modeling
 - Normalization
 - Problems with un-normalized database
 - Normal Forms
 - Benefits of Normalization
 - Drawbacks of Normalization



Review Question

- Question 1: The higher the NF applicable to a table, the less vulnerable it is to inconsistencies and anomalies.
 - True/False
- Question 2: _____ remove partial key dependencies



DBMS SQL

Lesson 08: Set Operators

Lesson Objectives

- To understand the following topics:
 - Set Operators
 - UNION operator
 - INTERSECT operator
 - MINUS operator



7.1: Set Operation

SET Operators in Oracle

- SQL supports the following four Set operations:
 - UNION ALL
 - Combines the results of two SELECT statements into one result set.
 - UNION
 - Same as UNION ALL. Eliminates duplicate rows from that result set.
 - MINUS
 - Takes the result set of one SELECT statement, and removes those rows that are also returned by a second SELECT statement.
 - INTERSECT
 - Returns only those rows that are returned by each of two SELECT statements.



Copyright © Capgemini 2015. All Rights Reserved 3

Set Operators:

There are situations when we need to “combine the results” from two or more SELECT statements. SQL enables us to handle these requirements by using “Set operations”.

The result of each SELECT statement can be treated as a set. SQL set operations can be applied on these sets to arrive at a final result.

SQL supports the following four Set operations:

- UNION ALL
- UNION
- MINUS
- INTERSECT

All set operators have equal precedence.

If a SQL statement contains multiple set operators, Oracle evaluates them from the left to right if there are no parentheses explicitly specifying another order.

contd.

SET Operators in Oracle

- Each of these operations combines the results of two SELECT statements into a single result.
- Note: While using SET operators, the column names from the first query appear in the result set.



Copyright © Capgemini 2015. All Rights Reserved 4

SQL statements containing the Set operators are referred to as “compound queries”. Each SELECT statement in a compound query is referred to as a “component query”.

Two SELECT statements can be combined into a compound query by a set operation only if they satisfy the following two conditions:

The “result sets” of both the queries must have the “same number of columns”.

The “datatype” of each column in the “second result set” must match the “datatype” of its corresponding column in the “first result set”.

For example: If component queries select character data, then the datatype of the return values are determined as follows:

If both queries select values of datatype CHAR, then the returned values have datatype CHAR.

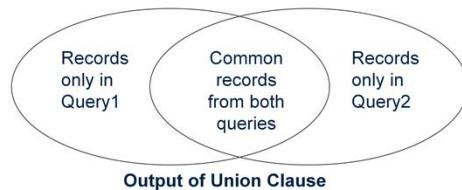
If either or both of the queries select values of datatype VARCHAR2, then the returned values have datatype VARCHAR2.

Tip: The datatypes do not need to be the same, if those in the second result set can be automatically converted by Oracle (using implicit casting) to types that are compatible with those in the first result set.

7.2: SET Operators

UNION Operator

- By using the UNION clause, multiple queries can be put together, and their output can be combined.
- The UNION clause merges the output of two or more queries into a single set of rows and columns.



Copyright © Capgemini 2015. All Rights Reserved 5

UNION Operator:

The UNION operator returns the records retrieved by either of the queries.

By default, the UNION operator eliminates duplicate records.

If however we want to retain duplicates, we use UNION ALL instead of UNION.

UNION operates over all of the columns being selected.

NULL values are not ignored during duplicate checking.

The IN operator has a higher precedence than the UNION operator.

By default, the output is sorted in ascending order of the first column of the SELECT clause.

UNION Operator- Example

Example: To display all students who are listed for 2006, 2007 and both the years

```
SELECT Student_Code FROM Student_Marks  
WHERE Student_year=2006  
UNION  
SELECT Student_Code FROM Student_Marks  
WHERE Student_year=2007;
```



Copyright © Capgemini 2015. All Rights Reserved 6

The query on the slide retrieves unique values from first query and second query. It also retrieves common values in both the queries by removing the duplicate records.

UNION Operator- Example

Some situations, if you need duplicate row as well use UNION ALL Operator

```
SELECT Student_Code FROM Student_Marks  
WHERE Student_year=2006  
UNION ALL  
SELECT Student_Code FROM Student_Marks  
WHERE Student_year=2007;
```



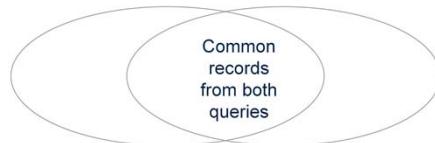
Copyright © Capgemini 2015. All Rights Reserved 7

The above query since is using UNION ALL will remove results returned by both the queries. The common values will be duplicate.

7.2: SET Operators

INTERSECT Operator

- The INTERSECT operator returns those rows, which are retrieved by both the queries.



Output of Intersect Clause



Copyright © Capgemini 2015. All Rights Reserved 8

INTERSECT Operator

INTERSECT result is same on reversing the order

INTERSECT does not ignore NULL values

INTERSECT Operator – Example

Example : To display students who are listed for both the years

```
SELECT Student_Code  
FROM Student_Marks WHERE Student_year=2006  
INTERSECT  
SELECT Student_Code  
FROM Student_Marks WHERE Student_year=2007;
```



Copyright © Capgemini 2015. All Rights Reserved 9

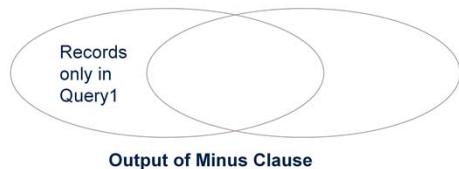
Example of INTERSECT operator:

The example on the slide will only display the common records retrieved by both the queries

7.2: SET Operators

MINUS Operator

- The MINUS operator returns all rows retrieved by the first query but not by the second query.



MINUS Operator - Example

Example: To display all students who are listed only for year 2006

```
SELECT Student_Code  
FROM Student_Marks WHERE Student_year=2006  
MINUS  
SELECT Student_Code  
FROM Student_Marks WHERE Student_year=2007;
```



Copyright © Capgemini 2015. All Rights Reserved 11

Example of MINUS operator:

The query on the slide will show results which are unique to first query

7.3: Tips and Tricks

Quick Guidelines

- Use UNION ALL in place of UNION.
 - The UNION clause forces all rows returned by each portion of the union to be sorted, merged and filtered for duplicates before the first row is returned to the "calling module".
 - A UNION ALL simply returns all rows including duplicates. It does not perform SORT, MERGE and FILTER.

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 12

Tips and Tricks:

When using the UNION statement, keep in mind, that the UNION statement performs the equivalent of a SELECT DISTINCT on the final result set, by default.

In other words, UNION takes the results of two like record sets, combines them, and then performs a SELECT DISTINCT in order to eliminate any duplicate rows.

This process occurs even if there are no duplicate records in the final record set.

Hence,

If you know that there are duplicate records, and it creates a problem for your application, then use the UNION statement "to eliminate the duplicate rows".

If you know that there will never be any duplicate rows, or if there are duplicates, and it does not create problems in your application, then you should use the UNION ALL statement instead of the UNION statement.

The advantage of the UNION ALL statement is that it does not perform the SELECT DISTINCT function. This saves a lot of server resources from being unnecessarily used.

Summary

- In this lesson you have learnt,
 - Use of Set Operations
 - Use of UNION operator
 - Advantage of UNION ALL over UNION operator
 - Use of INTERSECT operator
 - Use of MINUS operator



Review – Questions

■ Question 1: The Set operation that will show all the rows from both the resultsets including duplicates is _____.

- Option 1: Union All
- Option 2: Union
- Option 3: Intersect
- Option 4: Minus



■ Question 2: The Intersect operator returns _____.

Review – Questions

- Question 3: The output of set operators shows the columns names from ____.
- Question 4: The Union ALL clause performs SORT, MERGE and FILTER to give the required output.
 - True / False



DBMS SQL

Lesson 09: Database Objects

Lesson Objectives

- To understand the following Database Objects:
 - Table
 - Index
 - Synonym
 - Sequence
 - View



Overview

- A database is a collection of structures with appropriate authorizations and accesses that are defined.
- The structures in the database like tables, indexes, etc. are called as objects in the database.
- All objects that belong to the same user are said to be the “schema” for the particular user.
- Information about existing objects can be retrieved from dba_user_all_objects.



Copyright © Capgemini 2015. All Rights Reserved. 3

Database Objects:

Following is a list of Database objects:

Tables

Views

Indexes

Clusters

Synonyms

Sequences

Procedures

Functions

Packages

Triggers .

Basic Data Types

- Given below are the basic Data Types:

Datatype	Description
CHAR(n)	Stores fixed length string. Maximum length = 2000 bytes For example: NAME CHAR(15)
VARCHAR2(n)	Stores variable length string. Maximum length = 4000 bytes For example: DESCRIPTION VARCHAR2(100)
LONG(n)	Stores variable length string . Maximum length = 2 GIGA bytes For example: SYNOPSIS LONG(5000)
NUMBER(p,s)	Stores numeric data . Range is 1E-129 to 9.99E125 Max Number of significant digits = 38 For example: SALARY NUMBER(9,2)
DATE	Stores DATE. Range from January 1, 4712 BC to December 31, 9999 AD. Both DATE and TIME are stored. Requires 7 bytes. For example: HIREDATE DATE
RAW(n)	Stores data in binary format such as signature, photograph. Maximum size = 255 bytes
LONG RAW(n)	Same as RAW. Maximum size = 2 Gigabytes



Copyright © Capgemini 2015. All Rights Reserved.

4

Basic Data Types supported in SQL:

Scalar Data Types:

The traditional Data Types are called “Scalar Data Types”. The slide discusses some of the Scalar Data Types.

Basic Data Types contd..

Datatype	Description
TIMESTAMP	Stores the time to be stored as a date with fractional seconds. Extension to the DATE datatype There are some variations of the data type

➤ TimeStamp Datatype variations

TIMESTAMP [(fractional_seconds_precision)]	By default
TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE	This variant includes a time zone as displacement value which is difference between local time and UTC
TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE	Includes a time zone displacement in its value. The server returns the data in the users local time zone



Copyright © Capgemini 2015. All Rights Reserved.

5

Basic Data Types supported in SQL:

TheTimeStamp data type is introduced in Oracle 9i. This data type provides support for time zones. It is an extension of the DATE datatype.

The slide lists the variations supported for TIMESTAMP datatype
TIMESTAMP [(fractional_seconds_precision)] – It stores the year, month & day of the date data type. In addition to that it also stores hour, minute, second and fractional second value.

TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE - This variant includes a time zone as displacement value which is difference between local time and UTC (Universally Coordinated Time)

TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE – this variant includes time zone displacement in its value. This displacement value is not stored as part of the column data. The server returns the data in users local session time zone

In each variant the fractional_second_precision specifies the number of digits in the fractional part of the second and can be a number in the range of 0 to 9. This is an optional value and the default value is 6.

Basic Data Types contd..

Datatype	Description
BLOB	Binary Large Object • Stores any kind of data in binary format. • Typically used for multimedia data such as images, audio, and video.
CLOB	Character Large Object • Stores string data in the database character set format. Used for large strings or documents that exclusively use the database character set. • Characters in the database character set are in a fixed width format.
NCLOB	National Character Set Large Object • Stores string data in National Character Set format. Used for large strings or documents in the National Character Set. • Supports characters of varying width format.
BFILE	External Binary File • A binary file stored outside the database in the host operating system file system, but accessible from database tables. • BFILEs can be accessed from your application on a read-only basis. Use BFILEs to store static data, such as image data, that does not need to be manipulated in applications. • Any kind of data, that is, any operating system file, can be stored in a BFILE. For example: You can store character data in a BFILE, and then load the BFILE data into a CLOB specifying the character set upon loading.



Copyright © Capgemini 2015. All Rights Reserved.

6

Basic Data Types supported in SQL:

Oracle8i and Oracle9i's support for LOB (Large Object) data types is preferred over Oracle7's support for LONG and LONG RAWs in the following ways:

LOB Capacity: With Oracle8 and Oracle8i, LOBs can store up to 4GB of data. This doubles the 2GB of data that LONG and LONG RAW data types can store.

Number of LOB columns in a table: An Oracle8, Oracle8i, or Oracle9i table can have multiple LOB columns. Each LOB column in the same table can be of a different type. In Oracle7 Release 7.3 and higher, tables are limited to a single LONG or LONG RAW column.

Random piece-wise access: LOBs support random access to data, but LONGs support only sequential access.

LOBs can also be object attributes.

8.1: Database Objects

Table

- Tables are objects, which store the user data.
- Use the CREATE TABLE statement to create a table, which is the basic structure to hold data.

For example:

```
CREATE TABLE book_master  
(book_code number,  
book_name varchar2(50),  
book_pub_year number,  
book_pub_author varchar2(50));
```



Copyright © Capgemini 2015. All Rights Reserved.

7

Creating tables is done with the create table command. You can add rows to a table with the INSERT statement, after creating a table.

The above create table command does the following:

Defines the table name

Defines the columns in the table and the datatypes of those columns

In above example, we create a table called BOOK_MASTER which has 4 columns. The first column and third column is defined as NUMBER datatype. This means we will be storing numbers in this column. The second and fourth columns are of VARCHAR2 datatype. We will be storing text data in these columns

Syntax:

```
CREATE TABLE table_name  
{  
{col_name.col_datatype [[CONSTRAINT  
const_name][[col_constraint]]]},...  
[table_constraint],...  
}  
[AS query]
```

If a table is created as shown in the slide, then there is no restriction on the data that can be stored in the table.

However, if we wish to put some restriction on the data, which can be stored in the table, then we must supply some "constraints" for the columns. We will see the Constraints as next topic.

8.2: Table

What is Data Integrity?

- Data Integrity:

- "Data Integrity" allows to define certain "data quality requirements" that must be met by the data in the database.
- Oracle uses "Integrity Constraints" to prevent invalid data entry into the base tables of the database.
 - You can define "Integrity Constraints" to enforce the business rules you want to associate with the information in a database.
 - If any of the results of a "DML statement" execution violate an "integrity constraint", Oracle rolls back the statement and returns an error.



Copyright © Capgemini 2015. All Rights Reserved.

8

Data Integrity: Integrity Constraint

An Integrity Constraint is a declarative method of defining a rule for a column of a table.

Example of Data Integrity:

Assume that you define an "Integrity Constraint" for the Staff_Sal column of the Staff_Master table.

This Integrity Constraint enforces the rule that "no row in this table can contain a numeric value greater than 10,000 in this column".

If an INSERT or UPDATE statement attempts to violate this "Integrity Constraint", Oracle rolls back the statement and returns an "information error" message.

Advantages

- Advantages of Integrity Constraints:

- Integrity Constraints have advantages over other alternatives. They are:
 - Enforcing "business rules" in the code of a database application.
 - Using "stored procedures" to completely control access to data.
 - Enforcing "business rules" with triggered stored database procedures.



Copyright © Capgemini 2015. All Rights Reserved. 9

Applying Constraints

- Constraints can be defined at
- Column Level

```
CREATE TABLE tablename  
(column datatype [DEFAULT expr] [column_constraint] ,  
.....)
```

- Table Level

```
CREATE TABLE tablename  
(column datatype,  
 column datatype  
.....  
[CONSTRAINT constraint_name] constraint_type (column,...))
```



Copyright © Capgemini 2015. All Rights Reserved 10

Applying Constraints:

In Oracle you can apply constraints

Column Level - References a single column and is defined within a specification for the owning column; can be any type of integrity constraint

Table Level - References one or more columns and is defined separately from the definitions of the columns in the table; can define any constraints except NOT NULL

8.2: Table

Types of Integrity Constraints

- Let us see the types of Data Integrity Constraints:
 - Nulls
 - Unique Column Values
 - Primary Key Values
 - Referential Integrity



Copyright © Capgemini 2015. All Rights Reserved.

11

Types of Data Integrity:

Oracle supports the following Integrity Constraints:

NOT NULL constraints for the rules associated with nulls in a column. "Null" is a rule defined on a single column that allows or disallows, inserts or updates on rows containing a "null" value (the absence of a value) in that column.

UNIQUE key constraints for the rule associated with unique column values. A "unique value" rule defined on a column (or set of columns) allows inserting or updating a row only if it contains a "unique value" in that column (or set of columns).

PRIMARY KEY constraints for the rule associated with primary identification values. A "primary key" value rule defined on a key (a column or set of columns) specifies that "each row in the table can be uniquely identified by the values in the key".

FOREIGN KEY constraints for the rules associated with referential integrity. A "Referential Integrity" rule defined on a key (a column or set of columns) in one table guarantees that "the values in that key, match the values in a key in a related table (the referenced value)". Oracle currently supports the use of FOREIGN KEY integrity constraints to define the referential integrity actions, including:

- update and delete No Action
- delete CASCADE
- delete SET NULL

CHECK constraints for complex integrity rules

8.2: Table

NOT NULL Constraint

- The user will not be allowed to enter null value.

For Example:

- A NULL value is different from a blank or a zero. It is used for a quantity that is "unknown".
- A NULL value can be inserted into a column of any data type.

```
CREATE TABLE student_master  
(student_code number(4) NOT NULL,  
dept_code number(4) CONSTRAINT dept_code_nn  
NOT NULL );
```



Copyright © Capgemini 2015. All Rights Reserved.

12

NOT NULL constraint:

Often there may be records in a table that do not have values for every field.

This could be because the information is not available at the time of the data entry or because the field is not applicable in every case.

If the column is created as NULLABLE, in the absence of a user-defined value the DBMS will place a NULL value in the column.

A NULL value is different from a blank or a zero. It is used for a quantity that is "unknown".

"Null" is a rule defined on a single column that allows or disallows, inserts or updates on rows containing a "null" value (the absence of a value) in that column.

A NULL value can be inserted into a column of any data type.

Principles of NULL values:

Setting a NULL value is appropriate when the "actual value" is unknown, or when a value is not meaningful.

A NULL value is not equivalent to the value of "zero" if the data type is number, and it is not equivalent to "spaces" if the data type is character.

A NULL value will evaluate to NULL in any expression

For example: NULL multiplied by 10 is NULL.

NULL value can be inserted into columns of any data type.

If the column has a NULL value, Oracle ignores any UNIQUE, FOREIGN KEY, and CHECK constraints that may be attached to the column.

8.2: Table

DEFAULT clause

- If no value is given, then instead of using a “Not Null” constraint, it is sometimes useful to specify a default value for an attribute.

For Example:

- When a record is inserted the default value can be considered.

```
CREATE TABLE staff_master(  
Staff_Code number(8) PRIMARY KEY,  
Staff_Name varchar2(50) NOT NULL,  
Staff_dob date,  
Hiredate date DEFAULT sysdate,  
.....)
```



Copyright © Capgemini 2015. All Rights Reserved. 13

8.2: Table

UNIQUE constraint

- The keyword UNIQUE specifies that no two records can have the same attribute value for this column.

For Example:

```
CREATE TABLE student_master  
(student_code number(4),  
student_name varchar2(30)  
CONSTRAINT stu_id_uk UNIQUE(student_code));
```



Copyright © Capgemini 2015. All Rights Reserved. 14

UNIQUE constraint:

The UNIQUE constraint does not allow duplicate values in a column.

If the UNIQUE constraint encompasses two or more columns, then two equal combinations are not allowed.

However, if a column is not explicitly defined as NOT NULL, then NULLS can be inserted multiple number of times.

A UNIQUE constraint can be extended over multiple columns.

A “unique value” rule defined on a column (or set of columns) allows inserting or updating a row only if it contains a “unique value” in that column (or set of columns).

8.2: Table

PRIMARY KEY constraint

- The Primary Key constraint enables a unique identification of each record in a table.

For Example:

```
CREATE TABLE Staff Master  
(staff_code number(6)  
CONSTRAINT staff_id_pk PRIMARY KEY  
staff_name varchar2(20)  
.....);
```



Copyright © Capgemini 2015. All Rights Reserved. 15

PRIMARY KEY constraint:

On a technical level, a PRIMARY KEY combines a UNIQUE constraint and a NOT NULL constraint.

Additionally, a table can have at the most one PRIMARY KEY.

After creating a PRIMARY KEY, it can be referenced by a FOREIGN KEY.

A “primary key” value rule defined on a key (a column or set of columns) specifies that “each row in the table can be uniquely identified by the values in the key”.

The example on the slide defines the primary key constraint at the column level. The same example is seen below with the constraint defined at table level

```
CREATE TABLE Staff Master  
(staff_code number(6)  
staff_name varchar2(20)  
.....)  
CONSTRAINT staff_id_pk PRIMARY KEY (staff_code)  
;
```

8.2: Table

CHECK constraint

- CHECK constraint allows users to restrict possible attribute values for a column to admissible ones.

For Example:

```
CREATE TABLE staff_master
( staff_code number(2),
  staff_name varchar2(20),
  staff_sal  number(10,2) CONSTRAINT staff_sal_min
               CHECK (staff_sal >1000),
  ....);
```



Copyright © Capgemini 2015. All Rights Reserved. 16

CHECK constraint:

A CHECK constraint allows to state a minimum requirement for the value in a column.

If more complicated requirements are desired, an INSERT trigger must be used.

8.2: Table

FOREIGN KEY constraint

- The FOREIGN KEY constraint specifies a “column” or a “list of columns” as a foreign key of the referencing table.
- The referencing table is called the “child-table”, and the referenced table is called “parent-table”.

For Example:

```
CREATE TABLE student_master  
(student_code number(6),  
 dept_code number(4) CONSTRAINT stu_dept_fk  
REFERENCES department_master(dept_code),  
 student_name varchar2(30));
```



Copyright © Capgemini 2015. All Rights Reserved. 17

FOREIGN KEY Constraint Keywords:

Given below are a few Foreign Key constraint keywords:

FOREIGN KEY: Defines the column in the child table at the table constraint level.

REFERENCES: Identifies the table and column in the parent table.

ON DELETE CASCADE: Deletes the dependent rows in the child table when a row in the parent table is deleted.

ON DELETE SET NULL: Converts dependent FOREIGN KEY values to NULL.

You can query the USER_CONSTRAINTS table to view all constraint definitions and names.

You can view the columns associated with the constraint names in the USER_CONS_COLUMNS view.

In EMP table, for deptno column, if we want to allow only those values that already exist in deptno column of the DEPT table, we must enforce what is known as REFERENTIAL INTEGRITY. To enforce REFERENTIAL INTEGRITY, declare deptno field of DEPT table as PRIMARY KEY, and deptno field of EMP table as FOREIGN KEY as follows

FOREIGN KEY Constraint Keywords:

In the given example, FOREIGN KEY has been declared as a Table constraint.

```
CREATE TABLE Dept
(
    Deptno      NUMBER(2) CONSTRAINT
    DEPTNO_P_KEY PRIMARY KEY,
    Dname       VARCHAR2(14)    NOT NULL,
    Loc         VARCHAR2(13)    NOT NULL
);
CREATE TABLE Emp
(
    Empno NUMBER(4) CONSTRAINT P_KEY
    PRIMARY KEY,
    Ename VARCHAR2(10)  CONSTRAINT
    ENAME_NOT_NULL NOT NULL,
    Deptno NUMBER(2),
    Job CHAR(9) CONSTRAINT
    JOB_ALL_UPPER
    CHECK (Job =UPPER(Job)),
    Hiredate DATE DEFAULT SYSDATE,
    CONSTRAINT DEPTNO_F_KEY FOREIGN
    KEY (Deptno)
        REFERENCES Dept(Deptno)
);
```

Creation of database objects: Tables (contd.):

A table can have a maximum of 1000 columns. Only one column of type LONG is allowed per table.

Table_name, col_name, const_name	A string upto 30 characters length. Can be made up to A-Z,0-9,\$,_,# Must begin with a non-numeric ORACLE data characters.
Col_datatype	One of the previously mentioned types.
Col_constraint	A restriction on the column can be of following types: PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY, CHECK, Can be named. Only one PRIMARY KEY is allowed per table.
Table_constraint	A restriction on single or multiple columns. The types are same as in col_constraint. (NULL constraint is not allowed here).
AS query	Query is an SQL statement using SELECT command. SELECT command returns the rows from tables. It is useful if the table being created is based on an existing table. New table need not have all the columns of the old table. Names of columns can be different. All or part of the data can be copied.

Any object created by a user is accessible to the user and the DBA only.

To make the object accessible to other users, the creator or the DBA must explicitly give permission to others.

8.2: Table

Create new table based on existing table

- Constraints on an “old table” will not be applicable for a “new table”.

```
CREATE TABLE student_dept117 AS  
SELECT student_code, student_name  
FROM student_master WHERE dept_code = 117
```



Copyright © Capgemini 2015. All Rights Reserved. 20

Creating new table based on an existing table:

The example in the slide shows how to create a new table based on an existing table.

When the new table will be created, it will contain student information from department 117.

Constraints on an old table will not be applicable for the new table except NOT NULL constraint.

8.2: Table

ALTER Table

- Given below is an example of ALTER TABLE:

```
ALTER TABLE table_name
    [ADD (col_name col_datatype col_constraint ...)]]
    [ADD (table_constraint)]
    [DROP CONSTRAINT constraint_name]
    [MODIFY existing_col_name new_col_datatype
        new_constraint new_default]
    [DROP COLUMN existing_col_name]
    [SET UNUSED COLUMN existing_col_name];
```



Copyright © Capgemini 2015. All Rights Reserved. 21

Examples of ALTER TABLE:

Table_name must be an existing table.

A column cannot be removed from an existing table by using ALTER TABLE.

The uses of modifying columns are:

- Can increase the width of a character column, any time.
- Can increase the number of digits in a number, any time.
- Can increase or decrease the number of decimal places in a number column, any time. Any reduction on precision and scale can be on empty columns only.
- Can add only NOT NULL constraint by using Column constraints. All other constraints have to be specified as Table constraints.

8.2: Table

ALTER Table – Add clause

- The “Add” keyword is used to add a column or constraint to an existing table.

For adding three more columns to the emp table, refer the following example:

```
ALTER TABLE Student_Master  
ADD (last_name varchar2(25));
```



Copyright © Capgemini 2015. All Rights Reserved. 22

ALTER TABLE – Add clause:

The ADD clause allows to add a column or constraint. You can also add multiple columns in one statement separated by comma.

A column with constraints can also be added as shown in the following example:

```
ALTER TABLE Department_Master  
ADD (dept_name varchar2(10) NOT NULL);
```

8.2: Table

ALTER Table – Add clause

- For adding Referential Integrity on "mgr_code" column, refer the following example:

```
ALTER TABLE staff_master  
ADD CONSTRAINT FK FOREIGN KEY (mgr_code) REFERENCES  
staff_master(staff_code);
```



Copyright © Capgemini 2015. All Rights Reserved. 23

8.2: Table

ALTER Table – MODIFY clause

- MODIFY clause:

- The “Modify” keyword allows making modification to the existing columns of a table.
 - For Modifying the width of “sal” column, refer the following example:

```
ALTER TABLE staff_master  
MODIFY (staff_sal number (12,2) );
```



Copyright © Capgemini 2015. All Rights Reserved.

24

ALTER TABLE- Modify Clause

The use of modifying the columns with the Enable | Disable clause are:

Can increase “column width” of a character any time.

Can increase the “number of digits” in a number at any time.

Can increase or decrease the “number of decimal places” in a number column at any time. Any reduction on “precision” and “scale” can only be on empty columns.

Can only add the NOT NULL constraint by using “column constraints”. Rest all other constraints have to be specified as “table constraints”.

8.2: Table

ALTER Table – Enable | Disable clause

- **ENABLE | DISABLE Clause:**

- The ENABLE | DISABLE clause allows constraints to be enabled or disabled according to the user choice without removing them from a table.
- Refer the following example:

```
ALTER TABLE staff_master DISABLE CONSTRAINT SYS_C000934;
```



Copyright © Capgemini 2015. All Rights Reserved.

25

8.2: Table

ALTER Table – DROP clause

- The DROP clause is used to remove constraints from a table.
 - For Dropping the FOREIGN KEY constraint on “department”, refer the following example:

```
ALTER TABLE student_master  
DROP CONSTRAINT stu_dept_fk ;
```



Copyright © Capgemini 2015. All Rights Reserved. 26

ALTER TABLE – Drop Clause

To remove the PRIMARY KEY constraint on the DEPARTMENT table and drop the associated FOREIGN KEY constraint on the DEPT_CODE column.

```
ALTER TABLE department_master  
DROP PRIMARY KEY CASCADE;
```

8.2: Table

Dropping Column

- Given below are the ways for “Dropping” a column:
 - 1a. Marking the columns as unused and then later dropping them.
 - 1b. The following command can be used later to permanently drop the columns.

```
ALTER TABLE staff_master SET UNUSED COLUMN staff_address;  
ALTER TABLE staff_master SET UNUSED (staff_sal, hiredate);
```

```
ALTER TABLE emp DROP UNUSED COLUMNS;
```



Copyright © Capgemini 2015. All Rights Reserved. 27

Ways for “Dropping” a column:

Marking the columns as “Unused” and then later dropping them:

Oracle onwards a new feature to “drop” and “set” the “unused columns” in a table is added

First command as shown in the slide, allows you to mark a column as unused and

Second command as shown in the following slide, lets you drop the unused column from the table to create more free space.

This feature drops the column from a table and releases any space back to the segment.

Note that:

Columns once marked as unused cannot be recovered.

Marking the columns as unused does not release the space occupied by them back to the database.

Until you actually drop these columns, they continue to count towards the absolute limit of 1000 columns per table.

If you mark a column of data type LONG as UNUSED, you cannot add another LONG column to the table until you actually drop the unused LONG column.

The advantage of the marking column as “unused” and then dropping them is that marking the columns is much faster process than dropping the columns.

You can refer to the data dictionary table USER_UNUSED_COL_TABS to get information regarding the tables with columns marked as unused.

8.2: Table

Dropping Column

- Directly dropping the columns.

```
ALTER TABLE staff_master DROP COLUMN staff_sal;
```



Copyright © Capgemini 2015. All Rights Reserved. 28

Ways for “Dropping” a column (contd.):

DROP COLUMN:

This feature allows directly dropping the column.

For DROP COLUMN command shown in the slide, to work successfully, the table should be exclusively locked by the user by giving the command.

All “indexes” defined on any of the target columns are also dropped.

All “constraints” that reference a target column are removed.

Note that this command should be used with caution.

The CASCADE CONSTRAINTS clause is used along with the DROP COLUMN clause.

The CASCADE CONSTRAINTS clause drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped columns.

The CASCADE CONSTRAINTS clause also drops all multicolumn constraints defined on the dropped columns

8.2: Table

Drop a Table

- The DROP TABLE command is used to remove the definition of a table from the database.

For Example:

```
DROP TABLE staff_master;  
  
DROP TABLE Department_master  
CASCADE CONSTRAINTS;
```



Copyright © Capgemini 2015. All Rights Reserved. 29

Deleting Database Objects: Tables

Deleting objects that exist in the database is an easy task. Just say:

```
DROP Obj_Type obj_name;
```

A table that is dropped cannot be recovered. When a table is dropped, dependent objects such as indexes are automatically dropped. Synonyms and views created on the table remain, but give an error if they are referenced.

You cannot delete a table that is being referenced by another table. To do so use the following:

```
DROP table-name CASCADE CONSTRAINTS;
```

8.2: Table

User_Tables & User_Objects

- To view the names of tables owned by the user, use the following query:
- To view distinct object types owned by the user, use the following query:

```
SELECT table_name  
FROM user_tables
```

```
SELECT DISTINCT object_type  
FROM user_objects;
```



Copyright © Capgemini 2015. All Rights Reserved. 30

8.3: Index

Usage of Index

- Index is a database object that functions as a “performance-tuning” method for allowing faster retrieval of records.
- Index creates an entry for each value that appears in the indexed columns.
- The absence or presence of an Index does not require change in wording of any SQL statement.



Copyright © Capgemini 2015. All Rights Reserved. 31

Index

An index

- Is a schema object
- Is used by the Oracle server to speed up the retrieval of rows by using a pointer
- Can reduce disk I/O by using a rapid path access method to locate data quickly
- Is independent of the table it indexes
- Is used and maintained automatically by the Oracle Server.
- Oracle uses indexes to avoid the need for large-table, full-table scans and disk sorts, which are required when the SQL optimizer cannot find an efficient way to service the SQL query

Usage of Index

- Syntax:

```
CREATE [UNIQUE] INDEX index_name  
ON table_name(col_name1 [ASC|DESC], col_name2,....)
```



Copyright © Capgemini 2015. All Rights Reserved. 32

Index:

An index creates an entry for each value that appears in the indexed columns (by default, Oracle creates B-tree indexes).

The absence or presence of an Index does not require change in wording of any SQL statement.

An Index is merely a fast access path to the data.

An Index only affects the speed of execution.

There are different types of Indexes, which can be created by the user.

A “Table” can have any number of “Indexes”.

An Index can be on a single column or on multiple columns.

In Oracle, up to 32 columns can be included in one Index.

Updation of all Indexes is handled by RDBMS.

UNIQUE ensures that all rows in a Table are unique.

An Index is in an ascending order, by default.

The RDBMS decides when to use the Index while accessing the data.

Removal of an Index does not affect the Table on which it is based.

When you create a PRIMARY or a UNIQUE constraint on the “Table”, a UNIQUE index is automatically created.

The name of the constraint is used for the Index name.

Creating an Index

Example 1: A simple example of an Index is given below:

```
CREATE INDEX staff_sal_index ON staff_master(staff_sal);
```

Example 2: To allow only unique values in the field “ename”, the CREATE statement should appear as shown below:

```
CREATE UNIQUE INDEX staff_ename_unindex  
ON staff_master(staff_name );
```



Copyright © Capgemini 2015. All Rights Reserved. 33

Note:

More Indexes on a Table does not mean faster queries.

Each DML operation that is committed on a Table with Indexes imply that the Indexes must be updated.

The more number of Indexes are associated with a Table, the more effort the Oracle server must make to update all the Indexes after a DML operation.

When do you create an Index?

You should create Indexes, only if:

the column contains a wide range of values

the column contains a large number of NULL values

one or more columns are frequently used together in a WHERE clause or join condition

the table is large and most queries are expected to retrieve less than 2–4% of the rows

If you want to enforce uniqueness, you should define a UNIQUE constraint in the Table definition. Then a “unique index” will be automatically created.

It is usually not worth creating an Index, if:

The Table is small.

The columns are not often used as a condition in the query.

Most queries are expected to retrieve more than 2 to 4 percent of the rows in the table.

The Table is frequently updated.

The indexed columns are referenced as part of an expression.

8.3: Index

How are Indexes created?

- Indexes can be either created “automatically” or “manually”.
 - Automatically: A unique Index is automatically created when you define a PRIMARY KEY or UNIQUE constraint in a table definition.
 - Manually: A non-unique index can be created on columns by users in order to speed up access to the rows.



Copyright © Capgemini 2015. All Rights Reserved. 34

Types of Indexes:

Two types of Indexes can be created.

One type of index is an “unique index”. The Oracle server automatically creates this index when you define a column in a table that has a PRIMARY KEY or a UNIQUE key constraint. The name of the Index is same as the name given to the constraint.

The other type of index is a “non-unique index”, which can be created by a user.

For example: You can create a FOREIGN KEY column index for a “join” in a query in order to improve retrieval speed.

Note: You can manually create a “unique index”. However, it is recommended that you create a “UNIQUE constraint”, which implicitly creates a “unique index”.

8.3: Index

Bitmap Index

In a Bitmap Index, a “bitmap” is used for each “key value” instead of a list of “rowids”.

- A regular index does not store columns that contain NULL.
- Oracle, allows you to create Bitmap indexes in which you can store columns containing NULL.

```
CREATE BITMAP INDEX staff_idx  
ON staff_master(design_code);
```



Copyright © Capgemini 2015. All Rights Reserved. 35

Bitmap Index:

The Bitmap Index is useful for some types of SQL statements like:

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

Bitmap Indexes are used to tune queries that use non-selective columns in their limiting conditions.

A Bitmap Index can have a maximum of 30 columns.

In Bitmap structures, a “two-dimensional array” is created with one column for every row in the table, which is indexed.

Each column represents a distinct value within the bitmapped index.
This two-dimensional array represents “each value within the index” multiplied by the “number of rows” in the table.

In the “row retrieval” stage, the Oracle decompresses the bitmap into the RAM data buffers, such that it can be rapidly scanned for matching values.

These matching values are delivered to Oracle in the form of a Row-ID list.

These Row-ID values may directly access the required information.

The real benefit of bitmapped indexing occurs when “one table” includes “multiple bitmapped indexes”.

Each individual column may have low cardinality.

The creation of multiple bitmapped indexes provides a very powerful method for rapidly answering difficult SQL queries.

8.3: Index

Function Based Index

➤ A Function-based Index computes the value of the “function” or “expression” and stores it in the Index.

➤ To create a Function-based Index:

- the user should have an “Query Rewrite” privilege, or
- the system parameter QUERY_REWRITE_ENABLE should be set to TRUE



Copyright © Capgemini 2015. All Rights Reserved. 36

Function-based Index:

Function-based Indexes allow creation of indexes in PL/SQL and Java, on their:

- Expressions
- Internal functions
- User-written functions

Function-based indexes ensure that the Oracle designer is able to use an index for its query.

Prior to Oracle8, the usage of a “built-in” function was not able to match the performance of an “Index”. Consequently, Oracle performed the dreaded full-table scan.

Examples of SQL with Function-based queries include the following:

```
Select * from customer where substr(cust_name,1,4) = 'BURL';
Select * from customer where to_char(order_date,'MM') = '01';
Select * from customer where upper(cust_name) = 'JONES';
Select * from customer where initcap(first_name) = 'Mike';
```

In Oracle, the dbms always interrogates the WHERE clause of the SQL statement to check the existence of a matching Index.

By using Function-based Indexes, the Oracle designer can create a matching index, which exactly matches the predicates within the SQL WHERE clause.

This ensures that the query is retrieved with a minimal amount of disk I/O at the fastest possible speed.

Creating an Index Examples

Example 1: The index shown below can be very useful when you query for comparing revenue - cost.

Example 2:

```
CREATE INDEX sales_margin_index  
    ON sales(revenue - cost )
```

```
CREATE INDEX uppercase_idx ON staff_master(UPPER(staff_name));
```



Copyright © Capgemini 2015. All Rights Reserved. 37

Note:

The USER_INDEXES data dictionary view contains the name of the index and its uniqueness.

The USER_IND_COLUMNS view contains the index name, the table name, and the column name.

8.4: Synonym

Usage of Synonym

- A “Synonym” is an “alias” that is used for any table, view, materialized view, sequence, procedure, function, or package.
- Since a Synonym is simply an alias, it does not require storage except for storage of its definition in the data dictionary.
- Synonyms are often used for “security” and “convenience”.
- Synonyms can be created as either “public” or “private”.
- Synonyms are useful in hiding ownership details of an object.



Copyright © Capgemini 2015. All Rights Reserved. 38

Synonym:

A Synonym simplifies access to objects. A Synonym is simply another name for an object.

With Synonyms, you can:

Ease referring to a table owned by another user.

Shorten lengthy object names.

Usage of Synonym

➤ Syntax

- where:
 - Existing_name is the name of a table, view, or sequence.
 - PUBLIC is used to grant permission to all users for accessing the object by using the new name. (This is done only by a DBA.)

```
CREATE [PUBLIC] SYNONYM another_name FOR existing_name
```



Copyright © Capgemini 2015. All Rights Reserved. 39

Creating a Synonym

- Here is an example for synonym:
 - Suppose a procedure "proc1" is created in a schema "scott". While calling this procedure, if the user refers it as "scott.proc1", then a synonym is created as:

```
Create synonym prc1 for scott.proc1;
```



Copyright © Capgemini 2015. All Rights Reserved. 40

Creating and Removing Synonyms:

You can create a shortened name for the SALGRADE table as shown in the following example:

```
CREATE SYNONYM s  
FOR salgrade  
Synonym Created.
```

You can create a shortened name for the DEPT_SUM_VU view as shown in the following example:

```
CREATE SYNONYM d_sum  
FOR dept_sum_vu;  
Synonym Created.
```

You can drop a Synonym as shown in the following example:

```
DROP SYNONYM d_sum;  
Synonym dropped.
```

8.5: Sequence

Usage of Sequence

- A “Sequence” is an object, which can be used to generate sequential numbers.
- A Sequence is used to fill up columns, which are declared as UNIQUE or PRIMARY KEY.
- A Sequence uses “NEXTVAL” to retrieve the next value in the sequence order.



Copyright © Capgemini 2015. All Rights Reserved. 41

Sequence:

A Sequence:

- automatically generates unique numbers.
- is a “sharable object”.
- is typically used to create a PRIMARY KEY value.
- replaces application code.
- speeds up the efficiency of accessing sequence values when cached in memory.

8.5: Sequence

Creating a Sequence

- For example, suppose we have created a sequence "seq_no", then its next value can be obtained as "seq_no.nextval".

```
CREATE SEQUENCE seq_name  
[INCREMENT BY n1] [START WITH n2]  
[MAXVALUE n3] [MINVALUE n4] [CYCLE|NOCYCLE]  
[CACHE|NOCACHE];
```



Copyright © Capgemini 2015. All Rights Reserved. 42

Sequence:

In the example shown in the slide:

START WITH indicates the first NUMBER in the series.
INCREMENT BY is the difference between consecutive numbers. If n1 is negative, then the numbers that are generated are in a descending order.
MAXVALUE and MINVALUE indicate the extreme values.
CYCLE indicates that once the extreme is reached, it starts the cycle again with n2.
NOCYCLE means that once the extreme is reached, it stops generating numbers.
CACHE caches the specified number of sequence values in the memory. This speeds access, but all cached numbers are lost when the database is shut down. The default value is 20

Confirming Sequences

As shown below, you need to verify your sequence values in the USER_SEQUENCES data dictionary table.

```
SELECT sequence_name, min_value, max_value,  
       increment_by, last_number  
FROM user_sequences;
```

The LAST_NUMBER column displays the next available sequence number if NOCACHE is specified.

Creating a Sequence

- Here is one more example of sequence:
 - s1 will generate numbers 1,2,3....,10000, and then stop.

```
CREATE SEQUENCE s1  
INCREMENT BY 1  
START WITH 1  
MAXVALUE 10000  
NOCYCLE ;
```



Copyright © Capgemini 2015. All Rights Reserved. 43

8.4: Sequence

NEXTVAL and CURRVAL pseudo columns

- NEXTVAL returns the next available sequence value.
 - It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for the Sequence before CURRVAL can be referenced.



Copyright © Capgemini 2015. All Rights Reserved. 44

Referencing a Sequence:

After you create a Sequence, it generates sequential numbers that can be used in your tables. You can reference the Sequence values by using the NEXTVAL and CURRVAL pseudocolumns.

NEXTVAL and CURRVAL Pseudocolumns:

The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified Sequence. You must qualify NEXTVAL with the sequence name. When you reference sequence.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL.

The CURRVAL pseudocolumn is used to refer a Sequence number that the current user has just generated.

NEXTVAL must be used to generate a sequence number in the session of the current user, before CURRVAL can be referenced.

You must qualify CURRVAL with the sequence name.

When “sequence.CURRVAL” is referenced, the last value returned to that user’s process is displayed.

8.4: Sequence

Characteristics of Sequence

- Characteristics of a Sequence:

- Caching the Sequence values in memory to give faster access to those Sequence values
- Gaps in Sequence values can occur when:
 - a rollback occurs
 - the system crashes
 - a Sequence is used in another table
- Viewing the next available value by querying the USER_SEQUENCES table, when the sequence is created with NOCACHE



Copyright © Capgemini 2015. All Rights Reserved.

45

By using a sequence you can Insert a new employee with empno 1 as shown below:
You can view the current value for the S1 sequence as shown below:

Usage of a Sequence:

Caching Sequence Values

Cache sequences in memory to provide faster access to those "sequence values".

The first time you refer the Sequence, the cache is populated.

Each request for the next Sequence value is retrieved from the cached sequence.

After the last sequence value is used, the next request for the Sequence pulls another cache of sequences into the memory.

Gaps in the Sequence

Although "sequence generators" issue "sequential numbers" without gaps, this action occurs independent of a commit or rollback. Therefore, if you roll back a statement containing a Sequence, the number is lost.

Another event that can cause gaps in the Sequence is a system crash. If the Sequence caches the values in memory, then those values are lost if there is a system crash.

Since Sequences are not directly tied to tables, the same Sequence can be used for multiple tables. If you do so, each table can contain gaps in the Sequential numbers.

Viewing the Next Available Sequence Value without Incrementing It

If the Sequence is created with NOCACHE, it is possible to view the next available Sequence value without incrementing it by querying the USER_SEQUENCES table.

8.4: Sequence

Drop a Sequence

- A Sequence can be removed from the data dictionary by using the DROP SEQUENCE statement.
- Once removed, the Sequence can no longer be referenced.

```
DROP SEQUENCE dept_deptid_seq;  
Sequence dropped.
```



Copyright © Capgemini 2015. All Rights Reserved. 48

Removing a Sequence

To remove a sequence from the data dictionary, use the DROP SEQUENCE statement.

To remove the Sequence, you must be the owner of the Sequence or possess the DROP ANY SEQUENCE privilege.

Syntax:

```
DROP SEQUENCE sequence;
```

where: sequence is the name of the sequence generator.

For more information, refer Oracle9i SQL Reference, “DROP SEQUENCE”.

contd.

Altering a Sequence

- To alter a Sequence use the following syntax:

```
ALTER SEQUENCE s1  
INCREMENT BY n1  
MAXVALUE n3  
CYCLE;
```

- s1 is an existing Sequence.
- To make the Sequence start from a new number, drop the Sequence and create it again.
- ALTER SEQUENCE has no effect on numbers that are already generated.

```
ALTER SEQUENCE s1  
INCREMENT BY 5  
MAXVALUE 25000  
CYCLE;
```

Guidelines for modifying Sequences:

Given below are a few guidelines for modifying Sequences:

- You must be the owner or have the ALTER privilege for the Sequence.
- Only future sequence numbers are affected.
- The Sequence must be dropped, and re-created to restart the Sequence at a different number.
- Some validation should be performed.

8.5: View

Usage of View

- A View can be thought of as a “stored query” or a “virtual table”, i.e. a logical table based on one or more tables.
 - A View can be used as if it is a table.
 - A View does not contain data.



Copyright © Capgemini 2015. All Rights Reserved. 48

Why do we use Views?

Views are used:

To restrict data access.

To make complex queries easy.

To provide data independence.

To present different views of the same data.

Features of Simple and Complex Views:

Features	Simple View	Complex View
Number of tables	One	One or more
Contains functions	No	Yes
Contains groups of data	No	Yes
DML operations through a View.	Yes	Not always

Usage of View

- Syntax

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view [(alias[, alias]...)]  
AS subquery  
[WITH CHECK OPTION [CONSTRAINT constraint]]  
[WITH READ ONLY [CONSTRAINT constraint]];
```



Copyright © Capgemini 2015. All Rights Reserved. 49

View:

Creating a View:

You can embed a sub-query within the CREATE VIEW statement.

The sub-query can contain complex SELECT syntax.

Characteristics of a View:

View is a logical table that is based on one or more Tables.

View can be used as if it is a Table.

View does not contain data.

Whenever a View is accessed, the query is evaluated. Thus a View is dynamic.

Any changes made in the View affects the Tables on which the View is based.

View helps to hide the following from the user:

- Ownership details of a Table, and

- Complexity of the query used to retrieve the data

"With check option" implies you cannot insert a row in the underlying Table, which cannot be selected by using the View.

If you use a ORDER BY clause in the View, then it automatically becomes a "Read-only View".

Understanding the syntax

OR REPLACE - re-creates the view if it already exists

FORCE - creates the view regardless of whether or not the base tables exist

NOCACHE - creates the view only if the base tables exist. This is default

View - is the name of the view

Alias - specifies names for the expressions selected by the view's query

Subquery - is a complete and valid SELECT statement

WITH CHECK OPTION - specifies that only rows accessible to the view can be inserted or updated

Constraint - is the name assigned to the CHECK OPTION

WITH READ ONLY - ensures that no DML operations can be performed on this view

8.5: View

Creating a View

- Given below is an example of a simple View:

```
CREATE VIEW staff_view  
AS  
SELECT * FROM staff_master  
WHERE hiredate >'01-jan-82';
```



Copyright © Capgemini 2015. All Rights Reserved.

51

Restrictions while using Views:

Updating / Inserting rows in the base table is possible by using Views, however, with some restrictions. This is possible only if the View is based on a single table.

The restrictions are :

Updation / Insertion not possible if View is based on two tables. However, this can be done in ORACLE 8 onwards.

Insertion is not allowed if the underlying table has any NOT NULL columns, which are not included in the View.

Insertion / Updation is not allowed if any column of the View referenced in UPDATE / INSERT contains "functions" or "calculations".

Insertion / Updation / Deletion is not allowed if View contains GROUP BY or DISTINCT clauses in the query.a

Creating a View

- Creating a Complex View:

- As shown in the example given below, create a Complex View that contains group functions to display values from two tables.

```
CREATE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
AS SELECT dept.dept_name, MIN(staff.staff_sal),
          MAX(staff. staff_sal), AVG(staff. staff_sal)
     FROM staff_master staff, department_master dept
    WHERE staff.dept_code = dept.dept_code
  GROUP BY dept.dept_name;
```



Copyright © Capgemini 2015. All Rights Reserved. 52

Modifying a View:

As shown in the example given below, modify the View "staff_vu" by using the CREATE OR REPLACE VIEW clause. Add an alias for each column name.

```
CREATE OR REPLACE VIEW staff_vu
  (id_number, name, sal, department_id)
AS SELECT
  staff_code, staff_name, staff_sal, dept_code
    FROM staff_master
   WHERE dept_code = 80;
```

Column aliases in the CREATE VIEW clause are listed in the same order as the columns in the sub-query.

Creating a View

- Creating a View with WITH CHECK OPTION:

```
CREATE VIEW staff_vw
AS
SELECT * FROM staff_master
WHERE deptno =10 WITH CHECK OPTION constraint cn;
```



Copyright © Capgemini 2015. All Rights Reserved.

53

WITH CHECK OPTION Specifies that only the rows accessible to the view can be inserted or updated. Constraint "cn" is the name assigned to the CHECK OPTION constraint in the above example. Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

A violation produces:

ORA-01402: view WITH CHECK OPTION where-clause violation

With the given error, the SQL tried to INSERT or UPDATE a record in a view that contained a WITH CHECK OPTION. The resulting INSERT or UPDATE violates the WHERE clause of the view.

Creating a Read-only View:

If a View is based on a single table, the user may manipulate records in the View, and it's underlying base table.

The WITH READ ONLY clause of the CREATE VIEW command can be used to prevent users from manipulating records in a View.

```
CREATE VIEW student_view
AS
SELECT * FROM student_master
WHERE hiredate >'01-jan-82' WITH READ
ONLY.
```

Rules for performing operation on View

- You can perform “DML operations” on simple Views.
- You cannot remove a row if the View contains the following:
 - Group functions
 - A GROUP BY clause
 - The DISTINCT keyword
 - The pseudocolumn ROWNUM keyword



Copyright © Capgemini 2015. All Rights Reserved. 54

Rules for performing DML Operations on a View:

You can perform “DML operations” on data “through a View” only if those operations follow certain rules.

You can remove a row from a view, unless it contains any of the following:

- Group functions
- A GROUP BY clause
- The DISTINCT keyword
- The pseudocolumn ROWNUM keyword

Removing a View:

You can remove a View by using the following syntax:

```
DROP VIEW viewname;
```

8.5: View

Inline View

- “Inline view” is not a schema object like a regular View. However, it is a temporary query with an alias.

```
SELECT dept_name,staff_name,staff_sal  
FROM staff_master staff,  
(SELECT dept_name,dept_code  
FROM department_master) dept  
WHERE staff.dept_code=dept.dept_code
```



Copyright © Capgemini 2015. All Rights Reserved. 55

Inline View:

The Inline View is a construct in Oracle SQL, where you can place a query in the SQL FROM clause, just as if the query was a Table name.

In the example shown in the slide, the portion of the query, which is highlighted, is the Inline view.

A new feature has been included from Oracle 8i. You can use ORDER BY clause in the Inline View. This is very useful when you want to find the top n values in a table.

contd.

Inline View

➤ You can use Order By clause, as well, in the Inline View.

- This is very useful when you want to find the top n values in a table.

```
SELECT rownum,staff_name  
FROM (SELECT staff_name,staff_sal  
      FROM staff_master ORDER BY staff_sal desc)  
WHERE rownum < 5
```



Copyright © Capgemini 2015. All Rights Reserved. 56

Inline View (contd.):

Common uses for Inline Views, in Oracle SQL, are:

- to simplify complex queries by removing “join” operations, and
- to condense several separate queries into a single query

In ANSI standard SQL, it is quite difficult to compare two result sets that are summed together in a single query.

This is a common problem with Oracle SQL, where specific values must be compared to a summary.

Without the use of an Inline View, several separate SQL queries will have to be written:

- one to compute the sums from each view, and
- another to compare the intermediate result sets

Deleting Database Objects

Use the following syntax for deleting database objects:

Example 1:

Given below is an example of deleting a table:

```
DROP Obj_Type obj_name;
```

```
DROP TABLE student_marks;
```



Copyright © Capgemini 2015. All Rights Reserved. 57

Examples:

A table, which is dropped, cannot be recovered.

Dependent objects such as Indexes are automatically dropped.

Synonyms and Views remain intact. However, they give an error when referenced.

Deleting a Database Objects

Example 2:

If new_emp is a Synonym for a table, then the Table is not affected in any way. Only the duplicate name is removed.

```
DROP SYNONYM new_emp;
```



Copyright © Capgemini 2015. All Rights Reserved 58

8.5: Tips and Tricks
Guidelines

- When creating tables based on subquery the number of specified columns if defined for the table should match to the number of columns in the subquery.
- Create an index if
 - A column contains a wide range of values
 - A column contains a large number of null values
 - One or more columns are frequently used together in a WHERE clause or a join condition
 - The table is large and most queries are expected to retrieve less than 2 to 4 percent of the rows



Copyright © Capgemini 2015. All Rights Reserved. 59

Guidelines

- An Index is not very useful if :
 - The table is small
 - The columns are not often used as a condition in the query
 - Most queries are expected to retrieve more than 2 to 4 percent of rows in the table
 - The table is updated frequently
 - The indexed columns are referenced as part of an expression



Copyright © Capgemini 2015. All Rights Reserved 60

Summary

- What are Database Objects?
- Basic Data Types
- Data Integrity
- Different types of Database Objects:
- Modification of Database Objects
- Deleting Database Objects



Review – Questions

- Question 1: Indexes can be created _____ or _____
- Question 2: _____ obtains the current sequence value
- Question 3: Synonyms can be created as either _____ or _____



Review – Questions

- Question 4: You cannot use ORDER BY clause, in the Inline View
 - True / False
- Question 5: Gaps in sequence values can occur when there is a rollback
 - True / False
- Question 3: Synonyms are useful in hiding ownership details of an object.
 - True / False



DBMS SQL

Lesson 10: Data Manipulation Language

Lesson Objectives

- To understand the following topics:
 - Concept of Data Manipulation Language
 - Inserting rows into a table
 - Deleting rows from a table
 - Updating rows in a table



9.1: Concept of Data Manipulation Language

Data Manipulation Language

- Data Manipulation Language (DML) is used to perform the following routines on database information:
 - Retrieve
 - Insert
 - Modify
- DML changes data in an object. If you insert a row into a table, that is DML.
- All DML statements change data, and must be committed before the change becomes permanent.



Copyright © Capgemini 2015. All Rights Reserved 3

9.2: Addition of Data into Tables

INSERT

- **INSERT command:**

- **INSERT** is a DML command. It is used to add rows to a table.
- In the simplest form of the command, the values for different columns in the row to be inserted have to be specified.
- Alternatively, the rows can be generated from some other tables by using a SQL query language command.



Copyright © Capgemini 2015. All Rights Reserved. 4

Addition of Data into Tables:

Requisites for using INSERT command:

- If values are specified for all columns in the order specified at creation, then col_names could be omitted.
- Values should match “data type” of the respective columns.
- Number of values should match the number of column names mentioned.
- All columns declared as NOT NULL should be supplied with a value.
- Character strings should be enclosed in quotes.
- Date values should be enclosed in quotes.
- Values will insert one row at a time.
- Query will insert all the rows returned by the query.
- The table_name can be a “table” or a “view”. If table_name is a “view”, then the following restrictions apply:
 1. The “view” cannot have a GROUP BY, CONNECT BY, START WITH, DISTINCT, UNION, INTERSECT, or MINUS clause or a join.
 2. If the “view” has WITH CHECK OPTION clause, then a row, which will not be returned by the view, cannot be inserted.

9.2: Addition of Data into Tables

Inserting Rows into a Table

- Inserting by specifying values:

Example: To insert a new record in the DEPT table

```
INSERT INTO table_name[(col_name1,col_name2,...)]  
{VALUES (value1,value2,...) | query};
```

```
INSERT INTO Department_master  
VALUES (10, 'Computer Science');
```



Copyright © Capgemini 2015. All Rights Reserved.

5

Inserting Rows into a Table:

Example:

- Inserting a row in EMP table giving all values.

```
INSERT INTO student_master  
VALUES(1001,'Amit',10,'11-Jan-80','Chennai');
```

- 10 is a dept number which exists in DEPARTMENT_MASTER table
- Inserting a row in STAFF_MASTER table giving some values.

```
INSERT INTO staff_master  
(staff_code,staff_name,design_code,dept_code)  
VALUES(100001,'Arvind',102,30);
```

- This row will be created if all the constraints like NOT NULL are satisfied.

Inserting Rows into a Table

- Inserting rows in a table from another table using Subquery:

Example: The example given below assumes that a new_emp_table exists. You can use a subquery to insert rows from another table.

```
INSERT INTO new_staff_table  
SELECT * FROM staff_master  
WHERE staff_master.hiredate > '01-jan-82';
```



Copyright © Capgemini 2015. All Rights Reserved. 6

Inserting Rows into a Table

➤ Inserting by using “substitution variables”:

Example: In the example given below, when the command is run, values are prompted every time.

```
INSERT INTO department_master  
VALUES (&dept_code, '&dept_name');  
Enter a value for dept_code : 20  
Enter a value for dept_name : Electricals
```



Copyright © Capgemini 2015. All Rights Reserved. 7

Inserting Rows into a Table:

Inserting by using “substitution variables”:

- The problem with the INSERT statement is that it adds only “one row” to the table.
- However, by using “substitution variables” the speed of data input can be increased.
- Whenever a “substitution variable” is placed in a “value” field, the user will be prompted to enter the “actual value” when the command is executed.

9.3: Deletion of Data from Tables

DELETE

- The DELETE command is used to delete one or more rows from a table.
- The DELETE command removes all rows identified by the WHERE clause.

```
DELETE [FROM] {table_name | alias }  
[WHERE condition];
```



Copyright © Capgemini 2015. All Rights Reserved. B

Deletion of Data from Tables

- The table_name can be a “table” or a “view”.
- The DELETE command is used to delete one or more rows from a table.
- The DELETE statement removes all rows identified by the WHERE clause.
 - This is another DML, which means we can rollback the deleted data, and that to make our changes permanent.
- If WHERE clause is omitted, all rows from the table are removed. Else all rows which satisfy the condition are removed.
- FROM clause can be omitted without affecting the statement.

Deleting Rows from Table

Example 1: If the WHERE clause is omitted, all rows will be deleted from the table.

Example 2: If we want to delete all information about department 10 from the Emp table:

```
DELETE FROM staff_master;
```

```
DELETE FROM student_master WHERE dept_code=10;
```



Copyright © Capgemini 2015. All Rights Reserved.

Deletion of Data from Tables

Example 3:

```
DELETE staff_master WHERE staff_name = 'Anil';
```

9.4: Modifying / Updating existing Data in a Table

UPDATE

- Use the UPDATE command to change single rows, groups of rows, or all rows in a table.
- In all data modification statements, you can change the data in only “one table at a time”.

```
UPDATE table_name  
SET col_name = value|  
    col_name = SELECT_statement_returning_single_value|  
    (col_name,...) = SELECT_statement  
[WHERE condition];
```



Copyright © Capgemini 2015. All Rights Reserved 10

Modifying / Updating existing Data in a Table:

- The table_name can be a “table” or a “view”.
- The “value” can be a value, an expression, or a query, which returns a single value.
- The UPDATE command provides automatic navigation to the data.
- **Note:** If the WHERE clause is omitted, all rows in the table will be updated by a value that is currently specified for the field. Else only those rows which satisfy the condition will be updated.

Updating Rows from Table

Example 1: To UPDATE the column “dname” of a row, where deptno is 10, give the following command:

```
UPDATE department_master  
SET dept_name= 'Information Technology'  
WHERE dept_code=10;
```



Updating Rows from Table

Example 2: To UPDATE the subject marks details of a particular student, give the following command:

```
UPDATE student_marks  
SET subject1= 80 , subject2= 70  
WHERE student_code=1005;
```



Copyright © Capgemini 2015. All Rights Reserved 12

Using a Subquery to do an Update

- For making salary of “Anil” equal to that of staff member 100006, use the following command:

```
UPDATE staff_master  
SET staff_sal = (SELECT staff_sal FROM staff_master  
                  WHERE staff_code = 100006 )  
WHERE staff_name = 'Anil';
```



Copyright © Capgemini 2015. All Rights Reserved 13.

MERGE statement

- The MERGE statement, provides the ability to conditionally update or insert data into a database table.
- The MERGE statement, performs an UPDATE if the row exists, and an INSERT if it is a new row:
 - Increases performance and ease of use
 - Is useful in data warehousing applications
 - Avoids separate updates



Copyright © Capgemini 2015. All Rights Reserved 14.

MERGE Statement

The MERGE statement is used mostly in data warehouse environments to build the data in warehouse

MERGE statement

- You can conditionally insert or update rows in a table by using the MERGE statement

```
MERGE INTO table_name table_alias
  USING (table|view|sub_query) alias
  ON (join condition) WHEN MATCHED THEN
    UPDATE SET
      col1 = col_val1,
      col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```



Copyright © Capgemini 2015. All Rights Reserved 15.

- INTO :** this is how we specify the target for the MERGE. The target must be either a table or an updateable view (an in-line view cannot be used here);
- USING :** the USING clause represents the source dataset for the MERGE. This can be a single table (as in our example) or an in-line view;
- ON () :** the ON clause is where we supply the join between the source dataset and target table. Note that the join conditions must be in parentheses;
- WHEN MATCHED:** this clause is where we instruct Oracle on what to do when we already have a matching record in the target table (i.e. there is a join between the source and target datasets). We obviously want an UPDATE in this case. One of the restrictions of this clause is that we cannot update any of the columns used in the ON clause (though of course we don't need to as they already match). Any attempt to include a join column will raise an unintuitive invalid identifier exception; and
- WHEN NOT MATCHED :** this clause is where we INSERT records for which there is no current match.
- Note that sqlplus reports the number of rows **merged**. This includes both the updates and inserts. Oracle treats MERGE as a MERGE and not an UPDATE+INSERT statement. The same is true of SQL%ROWCOUNT in PL/SQL.

Example on Merge

Example

```
CREATE table staff_copy as select staff_code,staff_name FROM  
staff_master where 1=2;  
  
MERGE into staff_copy using staff_master  
ON (staff_master.deptno=staff_copy.deptno)  
WHEN MATCHED THEN  
UPDATE SET staff_code=staff_master.staff_code,  
staff_name=staff_master.staff_name  
WHEN NOT MATCHED THEN  
INSERT (staff_code,staff_name) values  
(staff_master.staff_code,staff_master.staff_name);
```



Copyright © Capgemini 2015. All Rights Reserved 16

Summary

- The concept of Data Manipulation Language
- Inserting rows into a table
- Deleting rows from a table
- Updating rows in a table
- Using Merge Statement



Review - Questions

- Question 1: Both TRUNCATE statement and DELETE without condition removes the entire data from a table
 - True/False
- Question 2: All DML statements are auto committed
 - True/False
- Question 3: In a transaction, DDL statement after DML statement commits the changes done by DML.
 - True/False



Review - Questions

- Question 4: Inserting rows in a table emp1 from another table can be done using ____.
 - Option 1: insert into emp1(t1) as select empno from emp
 - Option 2: insert into emp1(t1) select empno from emp
 - Option 3: insert into emp1(t1) as select * from emp



DBMS SQL

Lesson 11: Transaction
Control Language

Lesson Objectives

- To understand the following topics:
 - Transactions
 - Statement execution
 - Transaction control
 - Commit Transactions
 - Commit Command
 - Rollback transactions
 - Save points



10.1: Introduction to Transactions

Defining Transaction

- A “transaction” is a logical unit of work that contains one or more SQL statements.
- “Transaction” is an atomic unit.
- The effects of all the SQL statements in a transaction can be either:
 - all committed (applied to the database), or
 - all rolled back (undone from the database)
- A “transaction” begins with the first executable SQL statement.



Copyright © Capgemini 2015. All Rights Reserved 3

Defining Transaction

- A “transaction” ends when any of the following occurs:
 - A user issues a COMMIT or ROLLBACK statement without a SAVEPOINT clause.
 - A user runs a DDL statement such as CREATE, DROP, RENAME, or ALTER.
 - If the current transaction contains any DML statements, Oracle first commits the transaction, and then runs and commits the DDL statement as a new, single statement transaction.
 - A user disconnects from Oracle. The current transaction is committed.
 - A user process terminates abnormally. The current transaction is rolled back.



Copyright © Capgemini 2015. All Rights Reserved -4

Note:

After one transaction ends, the next executable SQL statement automatically starts the subsequent transaction.

Statement Execution and Transaction Control

- A “SQL statement” that runs successfully is different from a committed transaction.
- However, until the “transaction” that contains the “statement” is committed, the “transaction” can be rolled back. As a result, all the changes in the statement can be undone.
- Hence we can say, “a statement, rather than a transaction, runs successfully”.



Copyright © Capgemini 2015. All Rights Reserved 5

Statement Execution and Transaction Control:

Executing successfully means that a single statement was:

Parsed

Found to be a valid SQL construction

Run without error as an atomic unit.

For example: All rows of a multi-row update are changed.

Commit Transactions

- Committing a transaction means making “permanent” all the changes performed by the SQL statements within the transaction.
 - This can be done either explicitly or implicitly.



Copyright © Capgemini 2015. All Rights Reserved 8

Note:

An “explicit request” occurs when the user issues a COMMIT statement.

An “implicit request” occurs after normal termination of an application or completion of a data definition language (DDL) operation.

The changes made by the SQL statement(s) of a transaction become permanent and visible to other users only after that transaction is committed. Queries that are issued after the transaction is committed will see the committed changes.

Statement-Level Rollback

If at any time during execution, a SQL statement causes an error, then all effects of the statement are rolled back.

The effect of the rollback is as if that statement had never been run. This operation is a statement-level rollback.

Errors discovered during SQL statement execution cause statement-level rollbacks.

For example: Attempting to insert a duplicate value in a primary key.

Single SQL statements involved in a deadlock (competition for the same data) can also cause a statement-level rollback.

Errors discovered during SQL statement parsing, such as a syntax error, have not yet been run, so they do not cause a statement-level rollback.

A SQL statement that fails causes a loss only of any work it would have performed by itself.

It does not cause the loss of any work that preceded it in the current transaction.

If the statement is a DDL statement, then the implicit commit that immediately preceded it is not undone.

Commit Transactions

- COMMIT statement makes “permanent” all the changes that are performed in the current transaction.
- Syntax:

```
COMMIT [WORK];
```



Copyright © Capgemini 2015. All Rights Reserved 8

Commit Transactions

- COMMIT types:

- Implicit: Database issues an implicit COMMIT before and after any data definition language (DDL) statement
- Explicit

Example of COMMIT command:

```
DELETE FROM student_master  
WHERE student_name = 'Amit';  
COMMIT ;
```



Copyright © Capgemini 2015. All Rights Reserved 9

Rollback Transactions

- Rolling back a transaction means “undoing changes” to data that have been performed by SQL statements within an “uncommitted transaction”.
 - Oracle uses “undo tablespaces” (or rollback segments) to store old values.
 - Oracle also uses the “redo log” that contains a record of changes.



Copyright © Capgemini 2015. All Rights Reserved 10

Rollback Transactions

- Oracle lets you roll back an entire “uncommitted transaction”.
- Alternatively, you can roll back the trailing portion of an “uncommitted transaction” to a marker called a “savepoint”.



Copyright © Capgemini 2015. All Rights Reserved 11

Types of Roll back:

All the following types of rollbacks use the same roll back procedure:

Statement-level rollback (due to statement or deadlock execution error)

Rollback to a savepoint

Rollback of a transaction due to user request

Rollback of a transaction due to abnormal process termination

Rollback of all outstanding transactions when an instance terminates abnormally

Rollback of incomplete transactions during recovery

In rolling back an entire transaction, without referencing any savepoints, there is an occurrence of the following sequence:

Oracle undoes all changes made by all the SQL statements in the transaction by using the corresponding undo tablespace.

Oracle releases all the locks of data for the transaction.

The transaction ends.

Savepoints in Transactions

- In a transaction, you can declare intermediate markers called "savepoints" within the context of a transaction.
- By using "savepoints", you can arbitrarily mark your work at any point within a long transaction.
- In this manner, you can keep an option that is available later to roll back the work performed, however:
 - before the current point in the transaction, and
 - after a declared savepoint within the transaction



Copyright © Capgemini 2015. All Rights Reserved 12

Savepoints in Transactions

- For example: You can use savepoints throughout a long complex series of updates. So if you make an error, you do not need to resubmit every statement.



Copyright © Capgemini 2015. All Rights Reserved 13

Savepoints in Transactions:

Savepoints are useful in application programs, as well. If a procedure contains several functions, then you can create a savepoint at the beginning of each function.

Then, if a function fails, it is easy:

to return the data to its state before the function began,
and
to re-run the function with revised parameters or perform a RECOVERY action.

After a rollback to a savepoint, Oracle releases the data locks obtained by rolled back statements. As a result:

Other transactions that were waiting for the previously locked resources can proceed.

Other transactions that want to update previously locked rows can do so.

When a transaction is rolled back to a savepoint, there is an occurrence of the following sequence:

Oracle rolls back only the statements run after the savepoint.

Oracle preserves the specified savepoint, but all savepoints that were established after the specified savepoint are lost.

Oracle releases all table and row locks acquired since that savepoint, however, it retains all data locks acquired previous to the savepoint.

The transaction remains active and can be continued.

Examples of Rollback and Savepoints

- Example 1:

```
INSERT INTO department_master  
VALUES (70, 'PERSONNEL') ;  
SAVEPOINT A ;  
INSERT INTO department_master  
VALUES (80, 'MARKETING') ;  
SAVEPOINT B ;
```

```
ROLLBACK TO A ;
```



Copyright © Capgemini 2015. All Rights Reserved 14

Examples of Rollback and Savepoints:

In the example in the above slide, after execution of 'ROLLBACK TO A' command, 'INSERT INTO department_master VALUES (70, 'PERSONNEL')' statement got committed (stored in a table permanently) and all other statements after the SAVEPOINT A, will get rolledback (undone).

Advantages of COMMIT and ROLLBACK statements:

With COMMIT and ROLLBACK statements, you can:

- ensure data consistency
- preview data changes before making changes permanent
- group logically related operations

State of the Data after COMMIT:

- Data changes are made permanent in the database.
- The previous state of the data is permanently lost.
- All users can view the results.
- Locks on the affected rows are released. Those rows are available for other users to manipulate.
- All savepoints are erased.

State of the Data after ROLLBACK:

DBMS discards all pending changes by using the ROLLBACK statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released

Syntax:

```
UPDATE...
COMMIT;
```

```
INSERT...
ROLLBACK;
```

Example of rolling back changes to a Marker:

- Create a marker in a current transaction by using the SAVEPOINT statement.
- Roll back to that marker by using the ROLLBACK TO SAVEPOINT statement.

```
UPDATE...
SAVEPOINT update_done;
Savepoint created.
INSERT...
ROLLBACK TO update_done;
Rollback complete.
```

Summary

- Transactions
 - Statement execution
 - Transaction control
 - Commit Transactions
 - Commit Command
 - Rollback transactions
 - Save points



Review – Questions

- Question 1 : ___ is a logical unit of work.
- Question 2: A transaction is committed when the user issues a DDL statement.
 - True/False
- Question 3: A transaction is rolled back when ____.
 - Option 1: rollback statement is issued
 - Option 2: the user session is abruptly terminated
 - Option 3: an error occurs in DML statement
 - Option 4: none of the above



DBMS/SQL

Lesson 12: Data Control
Language

Lesson Objectives

- Data Control Language
 - Grant
 - Revoke



11.1: Data Control Language

User Access Control

- With Oracle server you can maintain database security, by:
 - Controlling database access
 - Giving access to specific objects in the database
 - Confirming given and received privileges with the Oracle data dictionary
 - Creating synonyms for database objects



Copyright © Capgemini 2015. All Rights Reserved. 3

In a typical multi-user environment, you need to maintain security of the database access and use. As far as Database Security is concerned it can be at System Level and Data Level. System Security concerns items in the database such as users, usage of disk space etc... Data Security would mean access and usage of database objects.

11.1: Data Control Language

Object Privileges

- Object Privileges are required to manipulate the content of the database objects
- Owner of the object has all the object privileges on that object.
- Owner can give or take out privileges on a particular object
- Object privileges can differ from object to object



Copyright © Capgemini 2015. All Rights Reserved. 4

Object Privilege:

The right to execute a particular SQL statement is known as a Privilege. The DBA is a high level user who has access to all the objects in the database.

Object Privilege is a right to manipulate the content of database objects. In other words, object privilege is right to perform a particular action on a specific table, view etc.. Different object privileges are available for different types of objects. The owner of the object automatically has all object privileges on that object.

Object Privileges

Object Privilege	Table	View	Sequence	Procedure
SELECT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
UPDATE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
DELETE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
INSERT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
ALTER	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	
INDEX	<input checked="" type="checkbox"/>			
REFERENCES	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
EXECUTE				<input checked="" type="checkbox"/>



Copyright © Capgemini 2015. All Rights Reserved. 5

Each object has a set of privileges which can be granted. The lists of privileges which can be granted on a particular object is shown on the slide.

You can refer to USER_TAB_PRIVS_REC table to know which object privileges are granted to the user. Similarly you can also refer to USER_TAB_PRIVS_MADE for knowing which object privileges are granted on user's objects.

Data Control Language - Syntax

- Granting Object Privileges

```
GRANT object_privileges{ALL |[(columnname)]} ON object  
TO {user|role|public} WITH GRANT OPTION
```

- Revoking Object Privileges

```
REVOKE {privilege,[privilege...]}{ALL} ON object  
{user,[user,...]}|role|public} [CASCADE CONSTRAINTS]
```



Copyright © Capgemini 2015. All Rights Reserved. 6

Data Control Language:

The syntax of both the Grant and Revoke statements is given on the slide. A user can give privileges using the GRANT statement

Understanding GRANT Statement:

Object_privileges : the right/privilege that is to be given

ALL : specifies/indicates all object privileges

columnname : specifies the column name on which privilege is to be given

ON object : the object on which privilege is to be given

TO user|role|public: The username or rolename to which the privilege is to be given. ALL grants privilege on the object to all users.

WITH GRANT OPTION : allows the grantee to grant the object privileges to other users and roles

To remove privileges the user can use REVOKE statement. This removes privileges that you mention from users specified and any other users to whom those privileges were granted through the WITH GRANT OPTION clause.

Data Control Language - Example

- Grant Query and Update privileges

```
GRANT SELECT ON student_master TO user1,user2;
```

```
GRANT UPDATE (subject1,subject2,subject3  
ON student_marks TO user1,user2;
```

- Grant privileges and allow to pass it on

```
GRANT SELECT, INSERT ON student_master  
TO user1  
WITH GRANT OPTION;
```



Copyright © Capgemini 2015. All Rights Reserved. 7

The owner of the object can grant access to all users using the Public keyword

In the following example all the users will be able to query student_master table

```
GRANT SELECT ON student_master  
TO public;
```

Data Control Language -Example

- Revoking UPDATE privileges from user2

```
REVOKE UPDATE on student_marks FROM user2;
```



Copyright © Capgemini 2015. All Rights Reserved. 8

You can also revoke multiple privileges from multiple users/roles.

Summary

- Data Control Language
 - Grant
 - Revoke



Summary



Copyright © Capgemini 2015. All Rights Reserved. 9

Add the notes here.

Review Questions

- Question 1: Object Privileges are same on all objects
 - True/False

- Question 2 : REFERENCES privilege can be granted on
 - Option 1 :Table
 - Option 2 : View
 - Option 3 : Sequence



Copyright © Capgemini 2015. All Rights Reserved. 10

Add the notes here.

Oracle for Developers (DBMS SQL)

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
05-Feb-2009	0.1D	Rajita Dhumal	Content Creation
09-Feb-2009		CLS team	Review
20-May-2011	2.0	Sathiabama R	Integration Refinements
23-May-2011	2.0	Anu Mitra	Review

Table of Contents

<i>Getting Started</i>	4
Overview	4
Setup Checklist for DBMS SQL	4
Instructions.....	4
Learning More (Bibliography if applicable).....	4
<i>Problem Statement / Case Study</i>	5
<i>Lab 1. Data Query Language</i>	7
1.1: Data Query Language	7
<i>Lab 2. Single Row Functions</i>	9
2.1: Single Row Functions:.....	9
2.2: Group Functions:	10
<i>Lab 3. JOINS AND SUBQUERIES</i>	12
3.1: Joins and Subqueries	12
3.2: Set Operators	14
<i>Lab 4. Database Objects</i>	16
4.1: Database Objects	16
<i>Lab 5. Data Manipulation Language</i>	21
5.1: Data Manipulation Language.....	21
<i>Lab 6. Transaction Control Language Statements</i>	23
6.1: Transaction Control Language Statements.....	23
<i>Appendices</i>	24
Appendix A: DBMS SQL Standards	24
Appendix B: Coding Best Practices	27
Appendix C: Table of Examples	29

Getting Started

Overview

This lab book is a guided tour for learning DBMS SQL. It comprises 'To Do' assignments. Follow the steps provided and work out the 'To Do' assignments.

Setup Checklist for DBMS SQL

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher (P166 recommended)
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)

Please ensure that the following is done:

- Oracle Client installed.
- Connectivity to Oracle Server

Instructions

- For all coding standards refer Appendix A. All lab assignments should refer coding standards.
- Create a directory by your name in drive <drive>. In this directory, create a subdirectory <DBMS SQL>_assgn. For each lab exercise create a directory as lab <lab number>.

Learning More (Bibliography if applicable)

NA

Problem Statement / Case Study

Table descriptions

Designation_Masters

Name	Null?	Type
Design_code	Not Null	Number(3)
Design_name		Varchar2(50)

Department_Masters

Name	Null?	Type
Dept_Code	Not Null	Number(2)
Dept_name		Varchar2(50)

Student_Masters Table

Name	Null?	Type
Student_Code	Not Null	Number(6)
Student_name	Not Null	Varchar2(50)
Dept_Code		Number(2)
Student_dob		Date
Student_Address		Varchar2(240)

Student_Marks

Name	Null?	Type
Student_Code		Number(6)
Student_Year	Not Null	Number
Subject1		Number(3)
Subject2		Number(3)
Subject3		Number(3)

Staff_Masters

Name	Null?	Type
Staff_code	Not Null	Number(8)
Staff_Name	Not Null	Varchar2(50)
Design_code		Number

Dept_code		Number
HireDate		Date
Staff_dob		Date
Staff_address		Varchar2(240)
Mgr_code		Number(8)
Staff_sal		Number (10,2)

Book_Masters

Name	Null?	Type
Book_Code	Not Null	Number(10)
Book_Name	Not Null	Varchar2(50)
Book_pub_year		Number
Book_pub_author	Not Null	Varchar2(50)

Book_Transactions

Name	Null?	Type
Book_Code		Number
Student_code		Number
Staff_code		Number
Book_Issue_date	Not Null	Date
Book_expected_return_date	Not Null	Date
Book_actual_return_date		Date

Lab 1. Data Query Language

Goals	<ul style="list-style-type: none"> Query the database . Usage of various operators in select statements. NVL Function.
Time	1 hr 30 min

1.1: Data Query Language

1. Retrieve the details (Name, Salary and dept code) of the employees who are working in department 20, 30 and 40.
2. List the details of the employees with user defined Column headers.
3. Display the code, subjects and total marks for every student. Total Marks will be calculated as Subject1+Subject2+Subject3. (Refer Student_Marks table)
4. List the details of the staff whose designations are either PROFESSOR or LECTURER.
5. List the code, name, and department number of the employees who have experience of more than 18 years.
6. List the name and Designations of the staff who have joined before Jan 2003.
7. List the name, designation, and income for 10 years of the employees who are working in departments 10 and 30.
8. List the name and experience (in years) of employees who are working as LECTURER.
9. Display name concatenated with dept code separated by comma and space. Name the column as 'Student Info'.
10. List the Name and Salary of the staff who are earning between 12000 and 25000. Sort them based on their salaries and name.

- 11.** Display employees who do not have manager.

- 12.** Write a query which will display name, department code and date of birth of all students who were born between January 1, 1981 and March 31, 1983. Sort it based on date of birth (ascending).

- 13.** Get the Department number, and sum of Salary of all non managers where the sum is greater than 20000.

- 14.** Display the details of books that have not been returned and expected return date was last Monday.

- 15.** Display the name and department code of students. If student does not belong to any department, display "No Department". Label the column as "Department". (Hint: Use NVL function)

- 16.** Display the name and salary of the staff. Salary should be represented as X. Each X represents a 1000 in salary.

Sample Output

JOHN	10000	XXXXXXXXXX
ALLEN	12000	XXXXXXXXXXXX

Lab 2. Single Row Functions

Goals	<ul style="list-style-type: none"> • Querying tables using single row functions • Querying tables using date functions • Querying tables using number functions • Querying tables using group functions
Time	1 hr 45 min

2.1: Single Row Functions:

1. Display name and date of birth of students where date of birth must be displayed in the format similar to "January, 12 1981" for those who were born on Saturday or Sunday.
2. Display each staff name and number of months they worked for the organization. Label the column as 'Months Worked'. Order your result by number of months employed. Round the number of months to closest whole number.
3. List the details of the employees, whose names start with 'A' and end with 'S'.
4. List the name and job of the employees whose names should contain N as the second or third character, and ending with either 'N' or 'S'.
5. Create a query which will display Staff Name, Salary of each staff. Format the salary to be 15 character long and left padded with '\$'.
6. List the names of the Employees having '_' character in their name.
7. List the details of the employees who have joined in December (irrespective of the year).
8. Write a query that displays Staff Name, Salary, and Grade of all staff. Grade depends on the following table.

Salary	Grade
Salary >=50000	A

Salary >= 25000 < 50000	B
Salary >= 10000 < 25000	C
OTHERS	D

- 9.** Display the Staff Name, Hire date and day of the week on which staff was hired. Label the column as DAY. Order the result by the day of the week starting with Monday.
- 10.** Show staff names with the respective numbers of asterisk from Staff table.
- 11.** Show staff names with the respective numbers of asterisk from Staff table except first and last characters. For example: KING will be replaced with K**G.
- 12.** Show all staffs who were hired in the first half of the month.
- 13.** Display the staff name, hire date and day of the week on which the staff joined. Order the results by the day of the week starting with Monday.
- 14.** Write a query to find the position of third occurrence of 'i' in the given word 'Mississippi'.
- 15.** Write a query to find the pay date for the month. Pay date is the last Friday of the month. Display the date in the format "Twenty Eighth of January, 2002". Label the heading as PAY DATE.

2.2: Group Functions:

- 16.** Display the Highest, Lowest, Total & Average salary of all staff. Label the columns Maximum, Minimum, Total and Average respectively. Round the result to nearest whole number.
- 17.** Edit the above query and display the same for each Department Name.
- 18.** Write a query to display number of people in each Department. Output should display Department Code, Department Name and Number of People.

- 19.** Determine the number of managers without listing them. Label the column as 'Total Number of Managers'.
- 20.** Display Manager Code, Manager Name and salary of lowest paid staff in that manager's team. Exclude any group where minimum salary is less than 10000. Order the result on descending order of salary.

Lab 3. JOINS AND SUBQUERIES

Goals	<ul style="list-style-type: none"> • Querying multiple tables using joins • Querying tables using subqueries • Querying tables using set operators
Time	2 hr 30 min

3.1: Joins and Subqueries

1. Write a query which displays Staff Name, Department Code, Department Name, and Salary for all staff who earns more than 20000.

2. Write a query to display Staff Name, Department Code, and Department Name for all staff who do not work in Department code 10 and have 'A' in their name.

3. Display Staff Code, Staff Name, Department Name, and his manager's number and name. Label the columns Staff#, Staff, Mgr#, Manager.

4. Create a query that will display Student Code, Student Name, Department Name, Subject1, Subject2, and Subject3 for all students who are getting 60 and above in each subject from department 10 and 20.

5. Create a query that will display Student Code, Student Name, Book Code, and Book Name for all students whose expected book return date is today.

6. Create a query that will display Staff Code, Staff Name, Department Name, Designation, Book Code, Book Name, and Issue Date. For only those staff who have taken any book in last 30 days.

7. Generate a report which contains the following information.

Staff Code Staff Name Designation Department Name

Department Head

For all staff excluding HOD (List should not contain the details of Department head).

8. Generate a report which contains the following information

Student Code	Student Name	Department Name	Total Marks
HOD Name			

Sort the output on Department Name and Total Marks.

9. Generate a report which contains the following information.

Staff Code, Staff Name, Designation, Department, Book Code, Book Name, Author, Fine

For the staff who have not return the book. Fine will be calculated as Rs. 5 per day.
Fine = 5 * (No. of days = Current Date – Expected return date).

10. List Staff Code, Staff Name, and Salary for those who are getting less than the average salary of organization.

11. List the Staff Code, Staff Name who are not Manager.

12. Display Author Name, Book Name for those authors who wrote more than one book.

13. Display Staff Code, Staff Name, and Department Name for those who have taken more than one book.

14. Display top ten students for a specified department. Details are:

Student Code, Student Name, Department Name, Subject1, Subject2, Subject3, Total.

15. Display the Staff Name, Department Name, and Salary for those staff who are getting less than average salary in their own department

16. Create a query that will display the Staff Name, Department Name, and all the staff that work in the same department as a given staff. Give the column as appropriate label.

17. List the Student Code, Student Name for that student who got highest marks in all three subjects in Computer Science department for current year.

- 18.** Display the Student Code, Student Name, and Department Name for that department in which there are maximum number of student are studying.
- 19.** Display Staff Code, Staff Name, Department Name, and Designation for those who have joined most recently.
- 20.** Display the Manager Name and the total strength of his/her team.

3.2: Set Operators

Observe the following queries and do the given assignments.

```
create table previous_products (
    pid int,
    name varchar(40),
    unit varchar(40),
    price int,
    stock int );
```

Example 1: Sample Code

```
create table current_products (
    pid int,
    name varchar(40),
    unit varchar(40),
    price int,
    stock int );
```

Example 2: Sample Code

```
insert into previous_products values(7,'Uncle Bob''s Organic Dried Pears','12 - 1 lb
pkgs.',30.00,15);
insert into previous_products values(8,'Northwoods Cranberry Sauce','12 - 12 oz
jars',40.00,6);
insert into previous_products values(1,'Chang','24 - 12 oz bottles',19.00,17);
insert into previous_products values(3,'Aniseed Syrup','12 - 550 ml
bottles',10.00,13);
insert into previous_products values(4,'Chef Anton''s Cajun Seasoning','48 - 6 oz
jars',22.00,53);
insert into previous_products values(5,'Chef Anton''s Gumbo Mix','36
boxes',21.35,0);
```

```
insert into previous_products values(6,'Grandma''s Boysenberry Spread','12 - 8 oz jars',25.00,120);
```

Example 3: Sample Code

```
insert into current_products values(7,'Uncle Bob''s Organic Dried Pears','12 - 1 lb pkgs.',30.00,15);
insert into current_products values(8,'Northwoods Cranberry Sauce','12 - 12 oz jars',40.00,6);
insert into current_products values(9,'Mishi Kobe Niku','18 - 500 g pkgs.',97.00,29);
insert into current_products values(10,'Ikura','12 - 200 ml jars',31.00,31);
insert into current_products values(11,'Queso Cabrales','1 kg pkg.',21.00,22);
insert into current_products values(5,'Chef Anton''s Gumbo Mix','36 boxes',21.35,0);
insert into current_products values(6,'Grandma''s Boysenberry Spread','12 - 8 oz jars',25.00,120);
```

Example 4: Sample Code

- 1.** Get the details of all products irrespective of the fact whether they are in previous set or current set.
- 2.** Get the details of all products along with the repetition of those that were present both in the previous and current sets.
- 3.** Get the details of only those products which were present in the previous set and are still continuing.
- 4.** Get the details of all obsolete products (no longer continued).

Lab 4. Database Objects

Goals	Following set of questions are designed to implement the following concepts <ul style="list-style-type: none"> • Creating Database objects like tables, views , etc • Modifying Database objects • Deleting Database objects • Usage of Data Dictionary tables
Time	4 hr 30 min

4.1: Database Objects

1. Create the Customer table with the following columns.

Customerid	Number(5)
CustomerName	Number(10)
Address1	Varchar2(30)
Address2	Varchar2(30)

2. Modify the Customer table CustomerName column of datatype with Varchar2(30). CustomerName should not accept Nulls.

3. Add the following Columns to the Customer table.

Gender	Varchar2(1)
Age	Number(3)
PhoneNo	Number(10)

4. Insert rows with the following data in to the Customer table.

Insert into customer values: (1000, 'Allen', '#115 Chicago', '#115 Chicago', 'M', '25, 7878776')

In similar manner, add the below records to the Customer table:

- 1000, Allen, #115 Chicago, #115 Chicago, M, 25, 7878776
- 1001, George, #116 France, #116 France, M, 25, 434524
- 1002, Becker, #114 New York, #114 New York, M, 45, 431525

5. Add the Primary key constraint for CustomerId with the name CustId_Prim.

6. Insert the row given below in the Customer table and see the message generated by the Oracle server.

1002, John, #114 Chicago, #114 Chicago, M, 45, 439525

7. Disable the constraint on CustomerId, and insert the following data:

- 1002, Becker, #114 New York, #114 New York , M, 45, 431525
- 1003, Nanapatekar, #115 India, #115 India , M, 45, 431525

8. Enable the constraint on CustomerId of the Customer table, and see the message generated by the Oracle server.

9. Drop the constraint CustId_Prim on CustomerId and insert the following Data. Alter Customer table, drop constraint Custid_Prim.

- 1002, Becker, #114 New York, #114 New York , M, 45, 431525, 15000.50
- 1003, Nanapatekar, #115 India, #115 India , M, 45, 431525, 20000.50

10. Delete all the existing rows from Customer table, and let the structure remain itself using TRUNCATE statement.

11. In the Customer table, add a column E_email.

12. Drop the E_email column from Customer table.

13. Add a new column EmailId to Customer table.

14. Mark EmailId column as unused before dropping it.

15. Drop the unused EmailId column from the Customer table.

16. Define the COMMENT 'Customers Details' for Customer table.

17. Use Data Dictionary USER_TAB_COMMENTS to view the created comment.

18. Define the COMMENT 'Personal Contact no' for the phoneno column of the Customer table.

19. Use Data Dictionary USER_COL_COMMENTS to view the created comment.

20. Create the Suppliers table based on the structure of the Customer table. Include only the CustomerId, CustomerName, Address1, Address2, and phoneno columns.

Name the columns in the new table as SuppID, SName, Addr1, Addr2, and Contactno respectively.

21. Drop the above table and recreate the following table with the name CustomerMaster.

CustomerId	Number(5) Primary key(Name of constraint is CustId_PK)
CustomerName	Varchar2(30) Not Null
Address1	Varchar2(30) Not Null
Address2	Varchar2(30)
Gender	Varchar2(1)
Age	Number(3)
PhoneNo	Number(10)

22. Create the AccountsMaster table with the following Columns. Use sequence to generate Account number

CustomerId	Number(5)
AccountNumber	Number(10,2) Primary key(Name of constraint is Acc_PK)
AccountType	Char(3)
LedgerBalance	Number(10,2) Not Null

23. Relate AccountsMaster table and CustomerMaster table through CustomerId column with the constraint name Cust_acc.

24. Insert the following rows to the CustomerMaster table:

- 1000, Allen, #115 Chicago, #115 Chicago, M, 25, 7878776
- 1001, George, #116 France, #116 France, M, 25, 434524
- 1002, Becker, #114 New York, #114 New York, M, 45, 431525

25. Modify the AccountMaster table with the Check constraint to ensure AccountType should be either NRI or IND.

26. Insert 5 rows into the AccountsMaster table:

27. Modify the AccountsMaster table keeping a Check constraint with the name Balance_Check for the Minimum Balance which should be greater than 5000.

28. Modify the AccountsMaster table such that if Customer is deleted from Customer table then all his details should be deleted from AccountsMaster table.

29. Create Backup copy for the AccountsMaster table with the name 'AccountDetails'.

30. Change the name of the AccountDetails table to 'BackUpTable' table.

31. Create a view 'Acc_view' with columns CustomerId, CustomerName, AccountNumber, AccountType, and LedgerBalance from AccountsMaster. In the view Acc_view, the column names should be CustomerCode, AccountHolderName, AccountNumber, Type, and Balance for the respective columns from AccountsMaster table.

32. Create a view on AccountsMaster table with name vAccs_Dtls. This view should list all customers whose AccountType is 'IND' and their balance amount should not be less than 10000. Using this view any DML operation should not violate the view conditions.



Hint: Use the With Check Option constraint.

33. Create a view accsvw10 which will not allow DML statement against it.

34. Display the department from Staff table which has the highest salary by using Inline View.

35. List the top two highest earning employees in each department.

- 36.** Create a Sequence with the name Seq_Dept on Deptno column of Dept table. It should start from 40 and stop at 200. Increment parameter for the sequence Seq_Dept should be in step of 10.
- 37.** Insert three sample rows by using the above sequence in Dept table.
- 38.** Alter the above specified sequence with an increment by 5.
- 39.** Drop the Seq_Dept sequence.
- 40.** Create a Unique index with the name No_Name on DeptNo and Dname of Dept table.
- 41.** Get information on the index No_Name from the Data Dictionary.
- 42.** Create public synonym synEmp for the EMP table.
- 43.** Get Information on synonym synEmp from the Data Dictionary.

Lab 5. Data Manipulation Language

Goals	Creating table to do the following DML operations <ul style="list-style-type: none"> • Insert Records • Delete Records • Update Records
Time	1 hr

5.1: Data Manipulation Language

1. Create Employee table with same structure as EMP table.

SQL>Create table employee as select * from emp where 1=3

SQL>desc employee

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(50)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO		NUMBER(2)

SQL>select * from employee

2. Write a query to populate Employee table using EMP table's empno, ename, sal, deptno columns.

SQL>select * from employee

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH				800		20
7499	ALLEN				1600		30

7521	WARD				125 0		30
7566	JONES				297 5		20
7654	MARTIN				125 0		30
7698	BLAKE				285 0		30
7782	CLARK				245 0		10
7788	SCOTT				300 0		20
7839	KING				500 0		10
7844	TURNER				150 0		30
7876	ADAMS				110 0		20
7900	JAMES				950		30
7902	FORD				300 0		20
7934	MILLER				130 0		10

14 rows selected.

3. Write a query to change the job and deptno of employee whose empno is 7698 to the job and deptno of employee having empno 7788.

4. Delete the details of department whose department name is 'SALES'.

5. Write a query to change the deptno of employee with empno 7788 to that of employee having empno 7698.

6. Insert the following rows to the Employee table through parameter substitution.

- 1000,Allen,Clerk,1001,12-jan-01, 3000, 2,10
- 1001,George,analyst,null, 08 Sep 92, 5000,0, 10
- 1002, Becker, Manager, 1000, 4 Nov 92, 2800,4, 20
- 1003, 'Bill', Clerk, 1002, 4 Nov 92,3000, 0, 20

Lab 6. Transaction Control Language Statements

Goals	<ul style="list-style-type: none">• Creating a transaction• Creating a savepoint• Roll back and commit the transaction to the savepoint
Time	30 min

6.1: Transaction Control Language Statements

1. Insert rows with the following data into the Customer table. 6000, John, #115 Chicago, #115 Chicago, M, 25, 7878776, 10000

- 6001, Jack, #116 France, #116 France, M, 25, 434524, 20000
- 6002, James, #114 New York, #114 New York, M, 45, 431525, 15000.50

Use parameter substitution.

2. Create a Savepoint named 'SP1' after third record in the Customer table .

3. Insert the below row in the Customer table.

6003, John, #114 Chicago, #114 Chicago, M, 45, 439525, 19000.60

4. Execute rollback statement in such a way that whatever manipulations done before Savepoint sp1 are permanently implemented, and the ones after Savepoint SP1 are not stored as a part of the Customer table.

Appendices

Appendix A: DBMS SQL Standards

Key points to keep in mind:

NA

How to follow DBMS SQL standards:

- Decide upon a database naming convention, standardize it across your organization and be consistent in following it. It helps make your code more readable and understandable.
- Tables:
 - Tables represent the instances of an entity. For example, you store all your customer information in a table. Here “customer” is an entity and all the rows in the customers table represent the instances of the entity “customer”. So name your table using the entity it represents, namely “Customer”. Since the table is storing “multiple instances” of customers, make your table name a plural word.
 - Keeping this in mind:
 - name your customer table as “Customers”
 - name your order storage table as “Orders”
 - name your error messages table as “ErrorMessages”
 - Suppose your database deals with different logical functions and you want to group your tables according to the logical group they belong to. It will help prefixing your table name with a two or three character prefix that can identify the group.

For example: Your database has tables which store information about Sales and Human resource departments, then you can name all your tables related to Sales department as shown below:

- SL_NewLeads
- SL_Territories
- SL_TerritoriesManagers

You can name all your tables related to Human resources department as shown below:

- HR_Candidates
- HR_PremierInstitutes
- HR_InterviewSchedules

- Prefix the table names with owner names, as this improves readability, and avoids any unnecessary confusion.
- Primary keys:
 - Primary key is the column(s) that can uniquely identify each row in a table. So just use the column name prefixed with “pk_” + “Table name” for naming primary keys.
 - Given below is an example of how we can name the primary key on the CustomerID column of Customers table:
 - pk_Customers_CustomerID
 - Consider concatenating the column names in case of composite primary keys.
- Foreign keys:
 - Foreign keys are used to represent the relationships between tables which are related. So a foreign key can be considered as a link between the “column of a referencing table” and the “primary key column of the referenced table”.
 - We can use the following naming convention for foreign keys:
 - fk_referencing table + referencing column_referenced table + referenced column
 - Based on the above convention, we can name the foreign key, which references the CustomerID column of the Customers table from the Order's tables CustomerID column as shown below:
 - fk_OrdersCustomerID_CustomersCustomerID
 - Foreign key can be composite too. In that case, consider concatenating the column names of referencing and referenced tables while naming the foreign key. This might make the name of the foreign key lengthy. However, you should not be worried about it, as you will never reference this name from your code, except while creating/dropping these constraints.
- Default and Check constraints:
 - Use the column name to which the defaults /check constraints are bound to. Prefix it with “def” and “chk” prefixes respectively for Default and Check constraints.
 - We can name the default constraint for OrderDate Column as def_OrderDate, and the check constraint for OrderDate column as chk_OrderDate.
- Do not use any reserved words for naming my database objects, as that can lead to some unpredictable situations.
- Do not depend on undocumented functionality. The reasons are:

- you will not get support, when something goes wrong with your undocumented code
- undocumented functionality is not guaranteed to exist (or behave the same) in a future release or service pack, thereby breaking your code
- Make sure you normalize your data at least till third normal form. At the same time, do not compromise on query performance. A little de-normalization helps queries perform faster.

Appendix B: Coding Best Practices

Given below are a few best practices in DBMS SQL:

- Do not use `SELECT *` in your queries. Always write the required column names after the `SELECT` statement, as shown below:
`SELECT CustomerID, CustomerFirstName, City`

This technique results in less disk IO, less network traffic, and hence better performance.

- Try to avoid wildcard characters at the beginning of a word while searching by using the `LIKE` keyword. This is because this arrangement results in an index scan, which is defeating the purpose of having an index. The following statement results in an index scan, while the second statement results in an index seek:

1. `SELECT LocationID FROM Locations WHERE Specialities LIKE '%apples'`
2. `SELECT LocationID FROM Locations WHERE Specialities LIKE 'A%s'`

Also avoid searching with not equals operators (`<>` and `NOT`) as they result in table and index scans. If you must do heavy text-based searches, consider using the Full-Text search feature of SQL Server for better performance.

- Use “Derived tables” wherever possible, as they perform better. Consider the following query to find the second highest salary from Employees table:

```
SELECT MIN(Salary)
FROM Employees
WHERE EmplID IN
(
    SELECT TOP 2 EmpID
    FROM Employees
    ORDER BY Salary Desc
)
```

The same query can be re-written by using a derived table as shown below, and it performs twice as fast as the above query:

```
SELECT MIN(Salary)
FROM
(
    SELECT TOP 2 Salary
    FROM Employees
    ORDER BY Salary Desc
) AS A
```

This is just an example. The results might differ in different scenarios depending upon the database design, indexes, volume of data, etc. So test all the possible ways a query can be written and go with the efficient one.

- Use char data type for a column, only when the column is non-nullable. If a char column is nullable, it is treated as a fixed length column in SQL Server 7.0+. So a char(100), when NULL, will eat up 100 bytes, resulting in space wastage. So use varchar(100) in this situation. Of course, variable length columns do have a very little processing overhead over fixed length columns. Carefully choose between char and varchar depending upon the length of the data you are going to store.
- Always use a column list in your INSERT statements. This helps in avoiding problems when the table structure changes (like adding a column).
- Do not use the column numbers in the ORDER BY clause as it impairs the readability of the SQL statement. Further, changing the order of columns in the SELECT list has no impact on the ORDER BY when the columns are referred by names instead of numbers. Consider the following example, in which the second query is more readable than the first one:

```
SELECT OrderID, OrderDate
FROM Orders
ORDER BY 2
```

```
SELECT OrderID, OrderDate
FROM Orders
ORDER BY OrderDate
```

Appendix C: Table of Examples

Example 1: Sample Code	14
Example 2: Sample Code	14
Example 3: Sample Code	15
Example 4: Sample Code	15