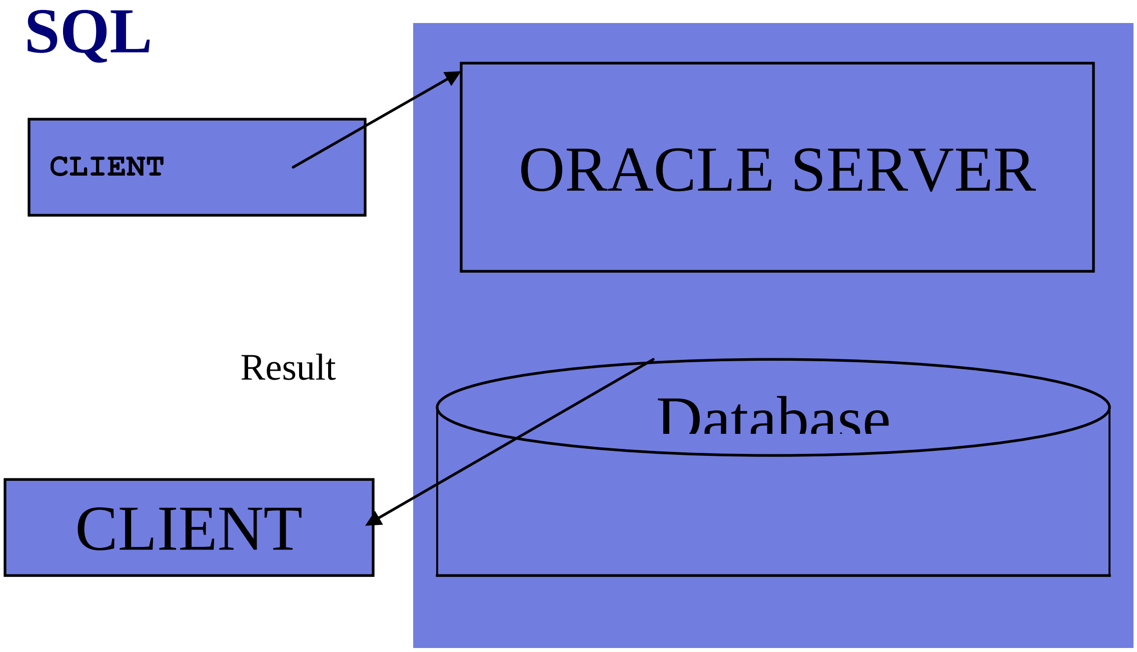
ORACLE

It is a DBMS, which manages a large amount of data in a multi user environment so that many users concurrently access the data. It also provides security and Recovery. It stores and manages data using relational model.

Oracle is the name of database management system developed by Oracle Corporation.

Oracle server manages data in the database. Users access Oracle server using SQL commands. So Oracle server receives SQL commands from users and executes them on the database.



Oracle's Role in Client / Server Computing

Client/Server computing is a method in which

- Database is stored on the server in the network
- A dedicated program, called back-end, runs on the server to manage database, which is also stored on the server.
- User access the data in database by running application, also called as front-end from clients, that accesses back-end running on the server.
- Applications running on the clients interact with the user.
- Back-end takes care of total database management.
- Client application and back-end run on different machines, which may be of different types. For example, back-end may run on mainframe and front-end may be on a PC.

Oracle is a database system that runs on the server, and used to manage the data. The other name to database server is Back-End.

The latest version of the Oracle server is 10g(for our training). Latest Version : 12i

Oracle server runs on different platforms. The following are some of the platforms on which Oracle runs.

- Windows NT.
- Novel Netware
- Unix

What is Personal Oracle?

Personal Oracle is one of the flavors of Oracle. In this Oracle server and client both run on the same machine. This is unlike other flavors where Oracle Server runs on Server and Front-end runs on Client.

It is possible to develop an application using Personal Oracle and deploy it in a Client / Server environment. Personal Oracle can support up to 15 database connections.

Features of Oracle

The following are some of the important features of Oracle Server.

Large Database Support

Oracle supports largest database, potentially hundreds of pita bytes in size. It also allows efficient usage of space by providing full control on space management.

Data Concurrence

Oracle supports concurrent access to database by multiple users. It automatically locks and unlocks rows to maintain integrity of the data.

Industry acceptance standards

Oracle server is 100% compliant with Entry of the ANSI / ISO standards. Oracle adheres to industry standards for data access language, network protocols etc. This makes Oracle an ‘open’ system, which protects the investment of customer. It is easy to port Oracle applications.

Portability

Oracle software can be ported to different operating systems and it is the name on all systems. Application development in Oracle can be ported to any operating system with little or no modifications.

Enforced Integrity

Oracle allows users to define business rules and enforce them. These rules need not be included at the application level.

Data Security

Oracle provides security in different levels – system level and object level. It also makes implementation of security easier through Roles.

Support for Client / Server environment

Oracle allows process to be split between client and server. Oracle server does all database management whereas Client does user interface. Oracle server allows code to be stored in the database in the form of procedures and functions. This allows centralization of the code and reduces network traffic.

Database Architecture

A database contains any length of information. But for the end user, we have to show only required information by hiding the unwanted information. This data hiding can be done using various data abstraction methods.

In any RDBMS we can use 3 levels of data abstractions.

- Physical level
- Logical Level
- View level

Physical Level

The Physical structure of the database is placed in Physical level. It is physically a set of three operating system files.

- Data Files
- Redo log files
- Control files

These files automatically create when database is created.

Data Files

It contains the data of the database. Every table that is stored in the database is a part of these files. Only Oracle Server can interpret these data files.

Redo Log Files

Every database has a set of two or more Redo Log files. The set of redo log files is known as databases redo log. Redo Log files are used in failure recovery. All changes made to the database are written to redo log file. (Filenames redo01.log)

Control Files

Contain information required to verify the integrity of the database, including the names of the other files in the database (Extension of file is ctl)

- Database Name
- Names and locations of data files and redo log files.

Path We can use this Oracle\oradata\orcl path in the server to see all the 3 types of files

Logical Structure

Logical Structure is independent of Physical structure. Each Oracle database contains the following components.

- Tablespaces
- Segments
- Extents
- Blocks

Tablespace

Each Database is a collection of tablespaces. For example we can use a table space called PAYROLL to store all the data related to payroll application.

Every database contains SYSTEM tablespace. This is automatically created when a database is created. SYSTEM tablespace contains the data dictionary tables.

It is possible to make tablespace temporarily unavailable by making it off-line and makes it available again by changing it to on-line. By making a tablespace off-line, DBA can take the backup.

Segments

Data into table spaces comes in the form of segments.
Example Table is a segment

An Oracle database requires up to 4 types of segments

➢ Data segments	It is used to store data of tables
➢ Index Segments	Used to store indexes
➢ Rollback segments	Undo information is stored
➢ Temporary segments	Oracle stores Temporary tables

Extents

The storage space is allocated to segments is in the form of Extents. Each Tablespace contains 65536 data files N number of such Table spaces creates a database. An extent is made with in a data file. N Number of continuous db blocks makes up an Extent.

Overall System Structure

A database system is partitioned into modules, which handles different responsibilities of over all system.

The functional components of a database system are

- Query processor Component
- Storage manager component

Query Processor Component

This component is a collection of the following processes.

- **DML Compiler** : It translates DML statements into a lower level instructions that the query evaluation engine understands

- **Embedded DML precompiler** It converts DML statements embedded in an application program into normal procedure calls in the host language.
- **DDL Interpreter** It interprets DDL statements and records them in a set of tables

- **Query evaluation engine** It executes lower level instructions generated by the DML compiler

Storage manager component

It is an Interface between the data stored in the database, application programs and queries submitted to the system.

- **Authorization and Integrity manager** It tests for satisfaction of integrity constraints and checks the authority of users to access data.
- **Transaction Manager** It ensures concurrent transaction executions processed without conflicting.
- **File manager** It manages the allocation of space on disk and the data structures used to represent information.
- **Buffer manager** This is responsible for fetching data from disk storage into main memory.

Oracle Instance

Every oracle database is associated with an Oracle Instance. Every time a database is started, a memory area called System Global Area (SGA) or Shared Global Area is allocated and one or more processes are started.

The combination of SGA and Oracle processes is called as Oracle Instance.

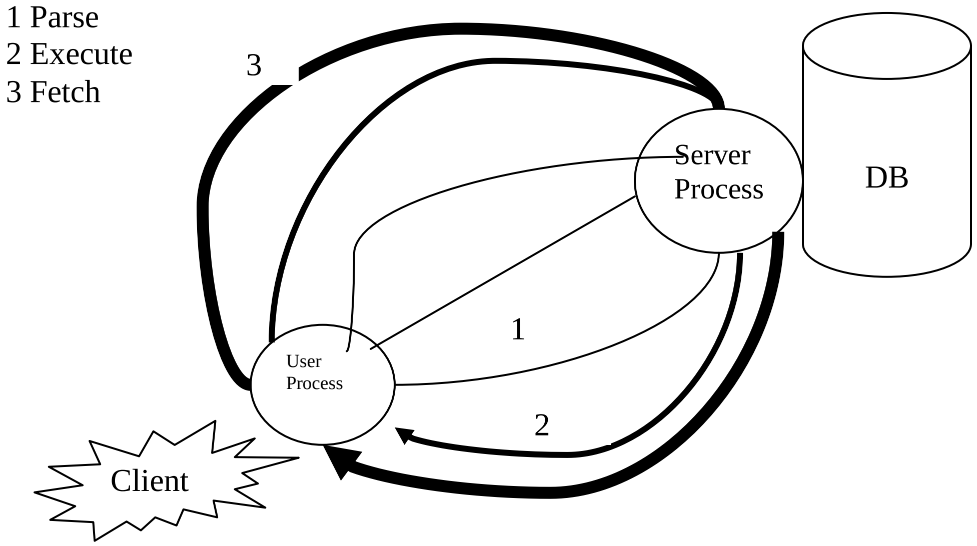
SGA consists of several memory structures:

- The shared pool is used to store the most recently executed SQL statements and the most recently used data from the data dictionary. These SQL statements may be submitted by a user process or, in the case of stored procedures, read from the data dictionary.

- The database buffer cache is used to store the most recently used data. The data is read from, and written to, the data files.

- The redo log buffer is used to track changes made to the database by the server and background processes.

Query process



When User connects to the database, it automatically creates two different processes called as User process and Server Process. The user process is the application program that originates SQL statements. The server process executes the statements sent from the user process.

There are three main stages in the processing of a query:

- Parse
- Execute
- Fetch

Parsing

During the *parse* stage, the SQL statement is passed from the user process to the server process, and a parsed representation of the SQL statement is loaded into a shared SQL area.
During the parse, the server process performs the following functions:

- Searches for an existing copy of the SQL statement in the shared pool
- validates the SQL statement by checking its syntax
- Performs data dictionary lookups to validate table and column definitions

Execute

Execute: Identify rows selected
The steps to be taken when executing the statement
The optimizer is the function in the Oracle Server that determines the optimal execution plan.

Fetch

Fetch: Return rows to user process. With each fetch process, it can fetch 20 records at a time.

DML Processing Steps

A data manipulation language (DML) statement requires only two phases of processing:

- Parse is the same as the parse phase used for processing a query
- Execute requires additional processing to make data changes

DML Execute Phase

The server process records the before image to the rollback block and updates the data block.
Both of these changes are done in the database buffer cache.

Any changed blocks in the buffer cache are marked as dirty buffers: that is, buffers that are not the same as the corresponding blocks on the disk.

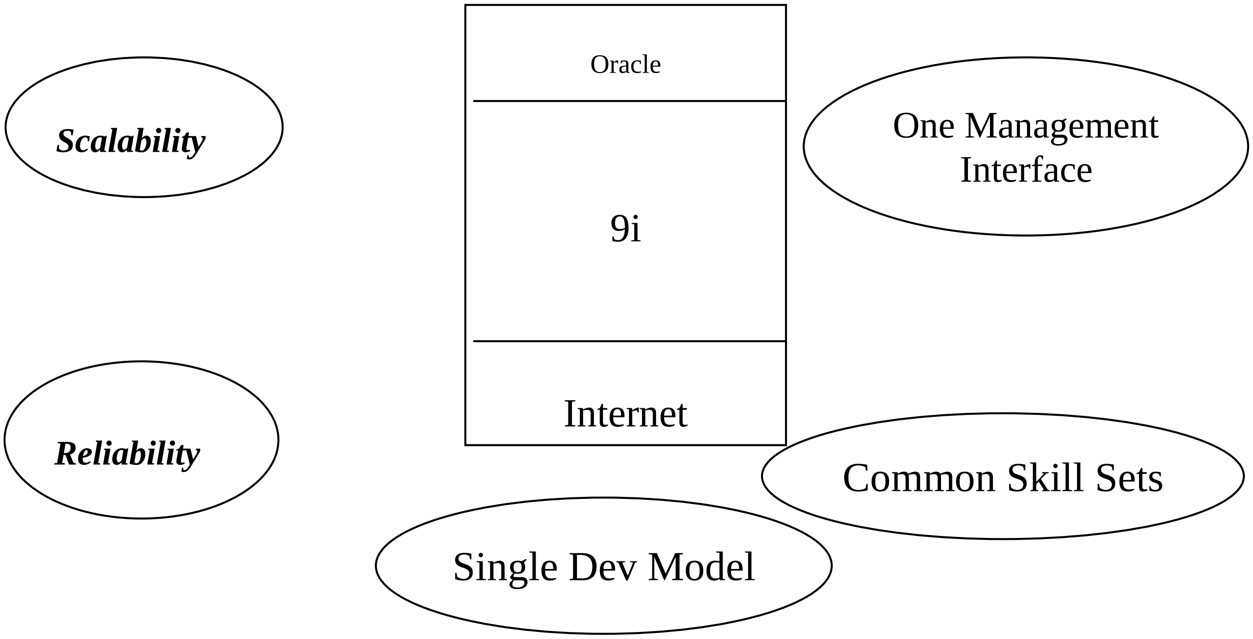
The processing of a `DELETE` or `INSERT` command uses similar steps. The before image for a `DELETE` contains the column values in the deleted row, and the before image of an `INSERT` contains the row location information.

Because the changes made to the blocks are only recorded in memory structures and are not written immediately to disk, a computer failure that causes the loss of the SGA can also lose these changes.

Oracle Versions

Oracle 6.0	1990
Oracle 7.0	1995
Oracle 7.1	1996
Oracle 7.2	1997
Oracle 7.3	1998 Object based
Oracle 8.0	1999 ORDBMS
Oracle 8i	2000 Internet based Application
Oracle9i	2001 Application server
Oracle10g	2004 Grid Computing
Oracle 11g	2009 482 new features
Oracle 12i	2009 Aug 1500 new features

Features & Benefits



Features

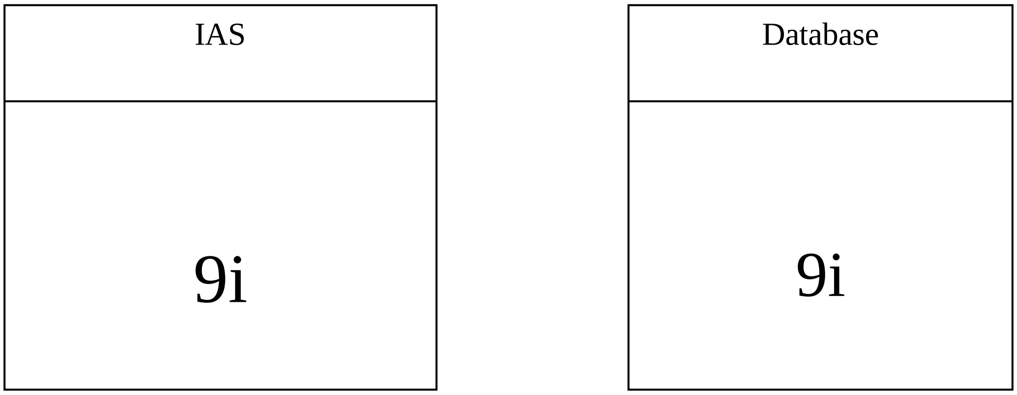
Oracle offers a comprehensive high performance infrastructure for e-business. It is called Oracle9i.It provides everything needed to develop, deploy and manage Internet applications.

Benefits

- Scalability from departments to enterprise e-business sites
- Reliable, available and secure architecture
- One development model, easy development options
- Common skill sets including SQL, PL/SQL,JAVA and XML
- One Management interface for all applications

9i Products

There are two products. They provide a complete and simple infrastructure for Internet applications.



Application Server

9i Application server runs all the applications and 9i database stores our data.
Oracle 9i Application server runs

- Portals or web sites
- Java Transactional Applications
- Provides integration between users, applications and data

Oracle9i: ORDBMS

Oracle is the first object-capable database developed by Oracle Corporation. It extends the data modeling capabilities of Oracle 8 to support a new object relational database model. Oracle provides a new engine that brings object-oriented programming, complex data types, complex business objects, and full compatibility with the relational world.

Oracle 9i supports

- User-Defined data types and objects
- Fully compatible with relational database (It supports all the CODD rules)
- Support of multimedia and Large objects
- It also support client server and web based applications

Oracle 9i can scale tens of thousands of concurrent users and support up to 512 peta bytes
of data (One peta byte = 1000 tera bytes and One terabyte = 1000 GB).

Data Concurrency And Consistency

Data concurrency allows many users to access the same data at the same time. One way of managing data concurrency is by making other users wait until their turn comes, when a user is accessing the data. But the goal of the database system is to reduce wait time so that it is negligible to each user. At the same time data integrity should not be compromised.

Oracle uses locking mechanism and multiversioning to increase data concurrency while maintaining data integrity.

Locking

Oracle uses Locks to control data concurrency. Locks are used to prevent destructive interference. For instance, when user X is modifying a row, it is locked so that other users cannot modify it until X completes modification. However, it doesn’t stop users querying the row. That means users reading the data will not be interrupted by user modifying and vice-versa.

Note: it is the responsibility of the application developer to unlock rows that are locked by committing or rolling back the transaction.

Read Consistency

For a query, Oracle returns a timepoint-based version of data. That means, the data retrieved is consistent with the time at which the query started. So any changes made to database since query started will not be available**.**

Ready consistency is made possible In Oracle using Rollback segment. Rollback segment keeps a copy of unchanged data. This data is substituted for the data that has changed since query started, in the query result.

This ensures readers do not wait for writers and vice-versa. And ensures that writers only wait for other writers, if they attempt to update identical rows at the same time.

Environment

Oracle uses two types of Environments for executing our SQL statements. SQL*plus and iSQL*plus.

ISQL*plus is (Available only from Oracle 9i)

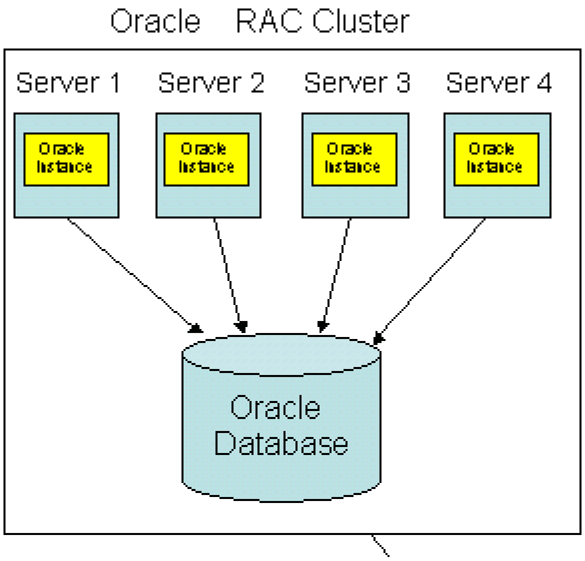
- An Environment
- Oracle proprietary
- Keywords can be abbreviated
- Runs on a browser
- Centrally loaded, does not have to be implemented on each machine

Difference between SQL*Plus and ISQL*plus

- SQL*Plus is a CUI and iSQL*Plus runs on a browser
- SQL*plus should be loaded in each every client system, where as iSQL*plus, centrally loaded, doesn't have to be implemented on each machine

Oracle 10g Grid Computing

1. Grid computing enables groups of networked computers to be pooled on demand to meet the changing needs of business.
2. Grid computing enables the creation of a single IT infrastructure that can be shared by multiple business processes.
3. Grid computing also employs special software infrastructure to monitor resource usage and allocate requests to the most appropriate resource.



Summary

Oracle is RDBMS. In a Client/Server environment, Oracle runs on the server as back-end to manage the data. The logical structure of the database is independent of physical structure of the database. User is concerned with only the logical structure of the database.

An Oracle Instance is the combination of SGA and Oracle process. Oracle instance contains a collection of background processes, where each process does a specific job.

Oracle uses locking mechanism to manage data concurrency. It copies the data of a row, before the row is changed, to rollback segment to provide read consistency.

Exercise

1) SGA stands for ______________________________
2) ____________ is the name of the tablespace that is automatically created when a database is created.
3) In which segment the data of a table is stored?
4) What is the difference between Personal Oracle and Client/Server Oracle
5) Redo Log files are also called as _____________
6) File extension of control file is __________
7) _______ Number of data files are there in each table space
8) What is Oracle Instance?
9) What is the maximum storage capacity of Oracle database
10) What is Application server?

Oracle Datatypes

Each column value and constant in a SQL statement has a data type, which is associated with a specific storage format, constraints, and a valid range of values. When you create a table, you must specify a data type for each of its columns. Oracle provides the following built-in data types.

- Character Data types
  - CHAR Data type
  - VARCHAR2 and VARCHAR Data types
  - NCHAR and NVARCHAR2 Data types

- LONG Data type
- NUMBER Data type
- DATE Data type
- LOB Data types
  - BLOB data type
  - CLOB and NCLOB data types
  - BFILE Data type

- RAW and LONG RAW Data types

Character Datatypes

The character data types store character (alphanumeric) data in strings, with byte values corresponding to the character.

CHAR datatype

Fixed length character data of length size in bytes.(Default size is 1 and maximum size is 2000).
Padded on right with blanks to full length of size.

VARCHAR2 (size)

Variable length characters strings having a maximum size of 4000 bytes (Default size is 1).
Truncates leftover blank spaces.

NVARCHAR2(size)

Variable length characters strings having a maximum size of 4000 bytes (Default size is 1)
Or characters, depending on the choice of national character set.
Truncates leftover blank spaces.

NUMBER(size,d)

For number column specified with **d** digits after the decimal point. For example, NUMBER (5,2) could nothing larger than 999.99 without an error.

LONG

Character data of variable size up to 2GB in length. Only one LONG column is allowed in a table.
Long column cannot be used in sub queries, functions, expressions, where clause or indexes.

DATE

Valid date ranges from January 1,4712 BC to December 31,9999 AD.
(Default date format DD-MON-YY)

TIMESTAMP(precision)

Date plus time, where precision is the number of digits in the fractional part of the seconds field (default is 6).

TIMESTAMP(precision) WITH TIME ZONE

Timestamp plus time zone displacement value.

TIMESTAMP(precision) WITHLOCAL TIME ZONE

TIMESTAMP, with normalized to local time zone.

RAW(size)

Raw binary date, size bytes long. Maximum size is 2000 bytes.

LONG RAW

Raw binary data, otherwise the same as LONG.

These two data types allow storing pictures.

CLOB
Character Large object, up to 4GB in length.

BLOB

Binary large object, up to 4GB in length.

BFILE
Pointer to a binary operating system file.

Introduction to SQL

A Brief History of SQL

The history of SQL begins in an IBM laboratory in San Jose, California, where SQL was developed in the late 1970s. The initials stand for Structured Query Language, and the language itself is often referred to as "sequel." It was originally developed for IBM's DB2 product (a relational database management system, or RDBMS, that can still be bought today for various platforms and environments). In fact, SQL makes an RDBMS possible. SQL is a nonprocedural language, in contrast to the procedural or third generation languages (3GLs) such as COBOL and C that had been created up to that time.

NOTE: Nonprocedural means what rather than how. For example, SQL describes what data to retrieve, delete, or insert, rather than how to perform the operation.

The characteristic that differentiates a DBMS from an RDBMS is that the RDBMS provides a set-oriented database language. For most RDBMS, this set-oriented database language is SQL. Set oriented means that SQL processes sets of data in groups.

Two standards organizations, the American National Standards Institute (ANSI) and the International Standards Organization (ISO), currently promote SQL standards to industry. The ANSI-92 standard is the standard for the SQL used throughout this book. Although these standard-making bodies prepare standards for database system designers to follow, all database products differ from the ANSI standard to some degree. In technology in a single-user business application positions the application for future growth.

An Overview of SQL

SQL is the standard language used to manipulate and retrieve data from these relational databases. SQL enables a programmer or database administrator to do the following:

- Modify a database's structure
- Change system security settings
- Add user permissions on databases or tables
- Query a database for information
- Update the contents of a database

SQL in Application Programming

SQL was originally made an ANSI standard in 1986. The ANSI 1989 standard (often called SQL-89) defines three types of interfacing to SQL within an application program:

Module Language-- Uses procedures within programs. These procedures can be called by the application program and can return values to the program via parameter passing.

Embedded SQL--Uses SQL statements embedded with actual program code. This method often requires the use of a precompiler to process the SQL statements. The standard defines statements for Pascal, FORTRAN, COBOL, and PL/1.

Direct Invocation--Left up to the implementer.

Before the concept of dynamic SQL evolved, embedded SQL was the most popular way to use SQL within a program. Embedded SQL, which is still used, uses *static* SQL—meaning that the SQL statement is compiled into the application and cannot be changed at runtime. The principle is much the same as a compiler versus an interpreter. The performance for this type of SQL is good; however, it is not flexible--and cannot always meet the needs of today's changing business environments.

Structured Query Language

Oracle server supports ANSI standard SQL and contains extensions. Using SQL, you can communicate with the Oracle server. SQL has the following advantages

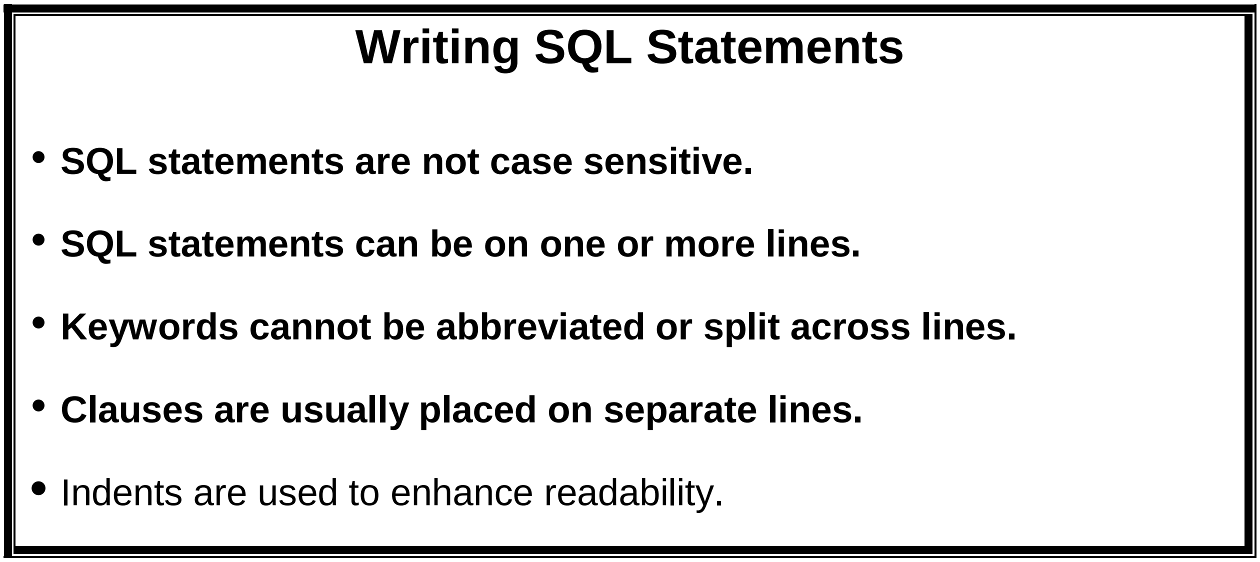
- Efficient
- Easy to learn and use
- Functionally complete (With SQL, you can define, retrieve, and manipulate data in the tables.)

SQL Statements

SELECT	Data retrieval

INSERT
UPDATE
DELETE
MERGE
	Data manipulation language(DML)

CREATE ALTER DROP RENAME TRUNCATE	Data definition language (DDL)
COMMIT ROLLBACK SAVEPOINT	Transaction control
GRANT REVOKE	Data control language (DCL)



Writing SQL Statements

- **SQL statements are not case sensitive.**
- **SQL statements can be on one or more lines.**
- **Keywords cannot be abbreviated or split across lines.**
- **Clauses are usually placed on separate lines.**
- Indents are used to enhance readability.

SELECT AND FROM

It is a building block for data retrieval in SQL.

Syntax : SELECT <COLUMNS> FROM <TABLE>;

Your First Query

INPUT:

```
SQL> select * from EMP;
```

OUTPUT:

```
EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
------ ---------- --------- ---------- --------- ---------- ---------- ----------
7369 SMITH      CLERK           7902 17-DEC-80        800                    20
7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
7566 JONES      MANAGER         7839 02-APR-81       2975                    20
7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
7788 SCOTT      ANALYST         7566 09-DEC-82       3000                    20
7839 KING       PRESIDENT            17-NOV-81       5000                    10
7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
7876 ADAMS      CLERK           7788 12-JAN-83       1100                    20
7900 JAMES      CLERK           7698 03-DEC-81        950                    30
7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
7934 MILLER     CLERK           7782 23-JAN-82       1300                    10
```

ANALYSIS:

Notice that columns 6 and 8 in the output statement are right justified and that columns 2 and 3 are left justified. This format follows the alignment convention in which numeric data types are right justified and character data types are left justified.

The asterisk (`*`) in `select *` tells the database to return all the columns associated with the given table described in the `FROM` clause. The database determines the order in which to return the columns.

A full table scan is used whenever there is no where clause on a query.

Terminating an SQL Statement

In some implementations of SQL, the semicolon at the end of the statement tells the interpreter that you are finished writing the query. For example, Oracle's SQL*PLUS won't execute the query until it finds a semicolon (or a slash). On the other hand, some implementations of SQL do not use the semicolon as a terminator. For example, Microsoft Query and Borland's ISQL don't require a terminator, because your query is typed in an edit box and executed when you push a button.

Changing the Order of the Columns

We can change the order of selection of columns

```
INPUT:
SQL> SELECT empno, ename, sal, job, comm from EMP;

OUTPUT

     EMPNO ENAME             SAL JOB             COMM
---------- ---------- ---------- --------- ----------
      7369 SMITH             800 CLERK
      7499 ALLEN            1600 SALESMAN         300
      7521 WARD             1250 SALESMAN         500
      7566 JONES            2975 MANAGER
      7654 MARTIN           1250 SALESMAN        1400
      7698 BLAKE            2850 MANAGER
      7782 CLARK            2450 MANAGER
      7788 SCOTT            3000 ANALYST
      7839 KING             5000 PRESIDENT
      7844 TURNER           1500 SALESMAN           0
      7876 ADAMS            1100 CLERK
      7900 JAMES             950 CLERK
      7902 FORD             3000 ANALYST
      7934 MILLER           1300 CLERK

14 rows selected.
```

Expressions, Conditions, and Operators

Expressions

The definition of an expression is simple: An *expression* returns a value. Expression types are very broad, covering different data types such as String, Numeric, and Boolean. In fact, pretty much anything following a clause (**SELECT** or **FROM**, for example) is an expression. In the following example **amount** is an expression that returns the value contained in the **amount** column.
SELECT sal FROM EMP;

In the following statement **NAME, DESIGNATION, SAL** are expressions:

SELECT ENAME, DESIGNATION, SAL FROM EMP;

Now, examine the following expression:

WHERE ENAME = 'KING'

It contains a condition, E**NAME = 'KING'**, which is an example of a Boolean expression. **ENAME = 'KING'** will be either **TRUE** or **FALSE**, depending on the condition **=**.

Conditions

If you ever want to find a particular item or group of items in your database, you need one or more conditions. Conditions are contained in the **WHERE** clause. In the preceding example, the condition is ENAME = 'KING'
To find everyone in your organization who worked more than **100** hours last month, your condition would be SAL > 2000
Conditions enable you to make selective queries. In there most common form, conditions comprise a variable, a constant, and a comparison operator. In the first example the variable is E**NAME**, the constant is **'KING'**, and the comparison operator is **=**.

In the second example the variable is **SAL**, the constant is **100**, and the comparison operator is **>**. You need to know about two more elements before you can write conditional queries: the **WHERE** clause and operators.

The WHERE Clause

The syntax of the **WHERE** clause is

SYNTAX:

```
SELECT <COLUMNS> FROM <TABLE> WHERE <SEARCH
CONDITION>;
```

SELECT, **FROM**, and **WHERE** are the three most frequently used clauses in SQL. **WHERE** simply causes your queries to be more selective. Without the **WHERE** clause, the most useful thing you could do with a query is display all records in the selected table(s).

If you wanted a particular EMPLOYEE, you could type

INPUT/OUTPUT:

```
SQL> SELECT * FROM EMP WHERE ENAME = 'KING';
```

Which would yield only one record:

```
EMPNO ENAME      JOB              MGR HIREDATE        SAL       COMM     DEPTNO
------ ---------- --------- ---------- --------- ---------- ---------- ----------
7839 KING       PRESIDENT            17-NOV-81      5000                     10
```

ANALYSIS:

This simple example shows how you can place a condition on the data that you want to retrieve.

If you wanted a particular EMPLOYEE, you could type

INPUT

```
SQL> SELECT * FROM BIKES WHERE ENAME != 'KING';
                OR
SQL> SELECT * FROM BIKES WHERE ENAME <> 'KING';
```

OUTPUT

```
EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
------- ---------- --------- ---------- --------- ---------- ---------- ----------
7369 SMITH      CLERK           7902 17-DEC-80       800                    20
7499 ALLEN      SALESMAN        7698 20-FEB-81      1600        300         30
7521 WARD       SALESMAN        7698 22-FEB-81      1250        500         30
7566 JONES      MANAGER         7839 02-APR-81      2975                    20
7654 MARTIN     SALESMAN        7698 28-SEP-81      1250       1400         30
7698 BLAKE      MANAGER         7839 01-MAY-81      2850                    30
7782 CLARK      MANAGER         7839 09-JUN-81      2450                    10
7788 SCOTT      ANALYST         7566 09-DEC-82      3000                    20
7844 TURNER     SALESMAN        7698 08-SEP-81      1500          0         30
7876 ADAMS      CLERK           7788 12-JAN-83      1100                    20
7900 JAMES      CLERK           7698 03-DEC-81       950                    30
7902 FORD       ANALYST         7566 03-DEC-81      3000                    20
7934 MILLER     CLERK           7782 23-JAN-82      1300                    10
```

ANALYSIS:

Displays all the employees other than KING.

Operators

Operators are the elements you use inside an expression to articulate how you want specified conditions to retrieve data. Operators fall into six groups: arithmetic, comparison, character, logical, set, and miscellaneous.

Arithmetic Operators

The arithmetic operators are plus (**+**), minus (-), divide (**/**), multiply (*****). The first four are self-explanatory. Modulo returns the integer remainder of a division.

Comparison Operators

True to their name, comparison operators compare expressions and return one of three values: **TRUE**, **FALSE**, or **Unknown**.

```
SELECT * FROM EMP WHERE SAL >= 2000;

SELECT * FROM EMP WHERE SAL >= 3000 AND SAL <= 4000;

SELECT * FROM EMP WHERE SAL BETWEEN 3000 AND 4000;

SELECT * FROM EMP WHERE SAL NOT BETWEEN 3000 AND
4000;
```

To understand how you could get an **Unknown**, you need to know a little about the concept of **NULL**. In database terms **NULL** is the absence of data in a field. It does not mean a column has a zero or a blank in it. A zero or a blank is a value. **NULL** means nothing is in that field. If you make a comparison like **Field = 9** and the only value for **Field** is **NULL**, the comparison will come back **Unknown**. Because **Unknown** is an uncomfortable condition, most flavors of SQL change **Unknown** to **FALSE** and provide a special operator, **IS NULL**, to test for a **NULL** condition.

Here's an example of **NULL**: Suppose an entry in the **PRICE** table does not contain a value for **WHOLESALE**. The results of a query might look like this:

```
SELECT * FROM EMP WHERE COMM IS NULL;


SELECT * FROM EMP WHERE COMM IS NOT NULL;
```

Character Operators

You can use character operators to manipulate the way character strings are represented, both in the output of data and in the process of placing conditions on data to be retrieved. This section describes two character operators: the **LIKE** operator and the || operator, which conveys the concept of character concatenation.

LIKE operator

What if you wanted to select parts of a database that fit a pattern but weren't quite exact matches? You could use the equal sign and run through all the possible cases, but that process would be time-consuming. Instead, you could use **LIKE**.

Consider the following:

INPUT:

```
SELECT * FROM EMP WHERE ENAME LIKE ‘A%’;
```

ANALYSIS

Displays all the employees whose names begins with letter A

INPUT:

```
SELECT * FROM EMP WHERE ENAME NOT LIKE ‘A%’;
```

ANALYSIS

Displays all the employees whose names not beginning with letter A

INPUT:

```
SELECT * FROM EMP WHERE ENAME  LIKE ‘%A%’;
```

ANALYSIS

Displays all the employees whose names contains letter A (Any number of A’s)

INPUT:

SELECT * FROM EMP WHERE ENAME LIKE ‘%A%A%’;

ANALYSIS

Displays all the names whose name contains letter A more than one time

INPUT:

SELECT * FROM EMP WHERE HIREDATE LIKE ‘%DEC%’;

ANALYSIS

Displays all the employees who joined in the month of December.

INPUT:

SELECT * FROM EMP WHERE HIREDATE LIKE ‘%81’;

ANALYSIS

Displays all the employees who joined in the year 81.

INPUT:

SELECT * FROM EMP WHERE SAL LIKE ‘4%’;

ANALYSIS

Displays all the employees whose salary begins with number 4.
(Implicit data conversion takes place).

Underscore (_)

The underscore is the single-character wildcard.

INPUT:

```
SQL> SELECT EMPNO,ENAME FROM EMP WHERE ENAME LIKE ‘_A%’;
```

OUTPUT:

```
     EMPNO ENAME
---------- ----------
      7521 WARD
      7654 MARTIN
      7900 JAMES
```

ANALYSIS

Displays all the employees whose second letter is A

INPUT:

```
SQL> SELECT * FROM EMP WHERE ENAME LIKE ‘__A%’;
```

OUTPUT:

```
ENAME
----------
BLAKE
CLARK
ADAMS
```

ANALYSIS

Displays all the employees whose third letter is A
(Two underscores followed by A)

INPUT:

```
SQL> SELECT * FROM EMP WHERE ENAME LIKE ‘A%\_%’ ESCAPE
‘\’;
```

OUTPUT:

```
ENAME
----------
AVINASH_K
ANAND_VARDAN
ADAMS_P
```

ANALYSIS

Displays all the employees with underscore (_). ‘\’ Escape character Underscore is used to identify a position in the string. To treat _ as a character we have to use Escape (\) character,

Concatenation (||) operator

Used to combine two given strings

INPUT

SELECT ENAME || JOB FROM EMP;

OUTPUT

```
ENAME||JOB
-------------------
SMITHCLERK
ALLENSALESMAN
WARDSALESMAN
JONESMANAGER
MARTINSALESMAN
BLAKEMANAGER
CLARKMANAGER
SCOTTANALYST
KINGPRESIDENT
TURNERSALESMAN
ADAMSCLERK
JAMESCLERK
FORDANALYST
MILLERCLERK
```

ANALYSIS

Combines both name and designation as a single string.

INPUT

SQL>SELECT ENAME || ‘ , ‘ || JOB FROM EMP;

OUTPUT

```
ENAME||','||JOB
----------------------
SMITH , CLERK
ALLEN , SALESMAN
WARD , SALESMAN
JONES , MANAGER
MARTIN , SALESMAN
BLAKE , MANAGER
CLARK , MANAGER
SCOTT , ANALYST
KING , PRESIDENT
TURNER , SALESMAN
ADAMS , CLERK
JAMES , CLERK
FORD , ANALYST
MILLER , CLERK
```

ANALYSIS

Combines both name and designation as a single string separated by,

Logical Operators

INPUT:

```
 SELECT ENAME FROM EMP WHERE ENAME  LIKE ‘%A%’ and
ENAME NOT LIKE ‘%A%A%’
```

OUTPUT

```
ENAME
----------
ALLEN
WARD
MARTIN
BLAKE
CLARK
JAMES
```

ANALYSIS

Displays all the employees whose names contains letter A exactly one time.

```
SELECT * FROM EMP WHERE SAL >= 3000 AND SAL <= 4000;

SELECT * FROM EMP WHERE SAL BETWEEN 3000 AND 4000;

SELECT * FROM EMP WHERE SAL NOT BETWEEN 3000 AND
4000;
```

Miscellaneous Operators: IN, BETWEEN and DISTINCT

The two operators **IN** and **BETWEEN** provide shorthand for functions you already know how to do. You could type the following:

INPUT:

```
SQL> SELECT ENAME, JOB FROM EMP WHERE JOB= 'CLERK'
 OR JOB =‘MANAGER’ OR JOB = 'SALESMAN';
```

OUTPUT:

```
ENAME      JOB
---------- ---------
SMITH      CLERK
ALLEN      SALESMAN
WARD       SALESMAN
JONES      MANAGER
MARTIN     SALESMAN
BLAKE      MANAGER
CLARK      MANAGER
TURNER     SALESMAN
ADAMS      CLERK
JAMES      CLERK
MILLER     CLERK
```

ANALYSIS

Display employees with designations manager, clerk, and salesman,

The above statement takes more time to parse it, which reduces the efficiency.

INPUT:

```
SQL> SELECT * FROM EMP WHERE JOB
IN('CLERK','SALESMAN','MANAGER');
```

OUTPUT:

```
ENAME      JOB
---------- ---------
SMITH      CLERK
ALLEN      SALESMAN
WARD       SALESMAN
JONES      MANAGER
MARTIN     SALESMAN
BLAKE      MANAGER
CLARK      MANAGER
TURNER     SALESMAN
ADAMS      CLERK
JAMES      CLERK
MILLER     CLERK
```

ANALYSIS

Display employees with designations manager, clerk, and salesman,

INPUT:

```
SQL> SELECT ENAME,JOB FROM EMP
 WHERE JOB NOT IN('CLERK','SALESMAN','MANAGER');
```

OUTPUT:

```
ENAME      JOB
---------- ---------
SCOTT      ANALYST
KING       PRESIDENT
FORD       ANALYST
```

ANALYSIS

Display designations other than manager, clerk, and salesman

INPUT:

```
SQL> SELECT ENAME,HIREDATE
 FROM EMP
 WHERE HIREDATE IN ('01-MAY-1981','09-DEC-1982');
```

OUTPUT:

```
ENAME      HIREDATE
---------- ---------
BLAKE      01-MAY-81
SCOTT      09-DEC-82
```

ANALYSIS

Display employees who joined on two different dates

DISTINCT OPERATOR

INPUT:

```
SQL>  SELECT DISTINCT JOB FROM EMP;
```

OUTPUT:

```
JOB
---------
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN
```

ANALYSIS

Distinct operator displays unique designations.

Distinct operator by default displays the information in ascending order.

ORDER BY CLAUSE

Display the information in a particular order (Ascending or descending order)

Syntax

SELECT <COLUMNS> FROM <TABLE> WHERE <CONDITION> ORDER BY <COLUMN(S)>;

INPUT

```
SQL> SELECT ENAME FROM EMP ORDER BY ENAME;
```

OUTPUT

```
ENAME
----------
ADAMS
ALLEN
BLAKE
CLARK
FORD
JAMES
JONES
KING
MARTIN
MILLER
SCOTT
SMITH
TURNER
WARD
```

ANALYSIS

Display employees in ascending order of names.

INPUT

```
SQL> SELECT JOB,ENAME,SAL FROM EMP ORDER BY JOB,ENAME;
```

OUTPUT

```
JOB       ENAME             SAL
--------- ---------- ----------
ANALYST   FORD             3000
ANALYST   SCOTT            3000
CLERK     ADAMS            1100
CLERK     JAMES             950
CLERK     MILLER           1300
CLERK     SMITH             800
MANAGER   BLAKE            2850
MANAGER   CLARK            2450
MANAGER   JONES            2975
PRESIDENT KING             5000
SALESMAN  ALLEN            1600
SALESMAN  MARTIN           1250
SALESMAN  TURNER           1500
SALESMAN  WARD             1250
```

ANALYSIS

Display employees in ascending order of jobs. With each job it places the information in ascending order of names.

INPUT

```
SQL> SELECT * FROM EMP ORDER BY job, name desc;
```

OUTPUT:

Display employees in ascending order by jobs. With each job it places the information in descending order of names.

INPUT

```
SQL> SELECT * FROM EMP ORDER BY job desc, ename desc;
```

OUTPUT:

Display employees in descending order by jobs. With each job it places the information in descending order of names.

INPUT

```
SQL> SELECT *
   FROM EMP where JOB != ‘CLERK’
  ORDER BY JOB;
```

OUTPUT:

Display employees in ascending order of jobs other than clerks.

We can also use order by clause as

INPUT

```
SQL> SELECT * FROM EMP ORDER BY 3;
```

ANALYSIS

It places the information in the order of third column in the table.

Exercise

1. _________ is the operator used to compare a column with null value.
2. _________is used to compare one value with a set of values.
3. The maximum number of character that can be stored in CHAR type is
4. Assume there is a table student(sname char(6), sname1 varchar2(6));
   Assume that value place in both columns is RAVI.
   What is the size of sname and sname1
5. How many LONG columns can a table contain?_______
6. SQL commands are to be terminated with ______________
7. Display list of employees that start with letter C
8. Display employees in ascending order of 5th column in the table

9. Examine the trace instance chart for employee table. You want to display each employee hiredate from earliest to latest. Which SQL statement will you use?

(1) SELECT hiredate FROM emp;

(2) SELECT hiredate FROM emp ORDER BY hiredate;

(3) SELECT emp FROM emp ORDER by hiredate;

(4) SELECT hiredate FROM emp ORDER BY hiredate DESC;

10. Which data type should you use for interest rates with varying and unpredictabledecimal places such as 1.234, 3.4, 5 and 1.23?

(1) LONG.

(2) NUMBER.

(3) NUMBER(p, s)

(4) None

11.Which SQL statement generates the alias Annual Salary for the calculated column SALARY*12?

(1) SELECT ename, salary*12 Annual SalaryFROM employees;

(2) SELECT ename, salary*12 Annual SalaryFROM employees;

(3) SELECT ename, salary*12 AS Annual SalaryFROM employees;

(4) SELECT ename, salary*12 AS INITCAP(ANNUAL SALARY)FROM employees

12. The EMP table has these columns: ENAME VARCHAR2(35), SALARY NUMBER(8,2)HIRE_DATE DATE. Management wants a list of names of employees who have been with the company for more than five years; Which SQL statement displays the required results?

   (1) SELECT ENAMEFROM EMPWHERE SYSDATE-HIRE_DATE > 5;

   (2) SELECT ENAMEFROM EMPWHERE HIRE_DATE-SYSDATE > 5;

   (3) SELECT ENAMEFROM EMPWHERE (SYSDATE-HIRE_DATE)/365 > 5;

   (4) SELECT ENAMEFROM EMPWHERE (SYSDATE-HIRE_DATE)* 365 > 5;

13. The employee table contains these columns.LAST_NAME VARCHAR2 (25), FIRST_NAME VARCHAR2(25) DEPT_ID NUMBER(9) You need to display the names of the employees that are not assigned to the department. Evaluate this SQL statement; SELECT last_name, first_name FROM employee WHERE dept_id is NULL which change should you make to achieve the desired result?

   (1) Create an outer join.

   (2) Change the column in the where condition.

   (3) Query executes successfully

   (4) Add a second condition to the where condition

14. Which statement about SQL is true?

   (1) Null values are displayed last in the ascending sequences.

   (2) Data values are displayed in descending order by default.

   (3) You cannot specify a column alias in an ORDER BY clause.

   (4) You cannot sort query results by a column that is not included in the SELECT list.

FUNCTIONS

A function is a sub program, which executes whenever we call it and returns a value to the calling place.

These functions are classified into two types

- Predefined functions
- User defined functions

Predefined functions

These functions are again classified into two types

- Group or Aggregate Functions
- Single row Functions

Aggregate Functions

These functions are also referred to as group functions. They return a value based on the values in a column.

COUNT

The function **COUNT** returns the number of rows that satisfy the condition in the **WHERE** clause.

Say you wanted to know how many employees are there.

```
INPUT:
SQL> SELECT COUNT(*) FROM EMP;

OUTPUT:

COUNT(*)
--------
14
```

ANALYSIS
It counts if row presents in the table

To make the code more readable, try an alias:

INPUT/OUTPUT:

```
SQL> SELECT COUNT(*) NUM_OF_EMP FROM EMP;

NUM_OF_EMP
-------------------
     14
```

INPUT/OUTPUT:

```
SQL> SELECT COUNT(COMM)
FROM EMP;

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

ANALYSIS

It counts only those when there is a value in comm. Column
Note: Count (*) faster than count(comm)
Count(*) count the row when a row present in the table where as Count(comm) counts the row only when there is a value in the

INPUT/OUTPUT:

```
SQL> SELECT COUNT(*) FROM EMP WHERE JOB = 'MANAGER';

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

ANALYSIS
It counts only managers

INPUT/OUTPUT:

```
SQL> SELECT count (distinct job) FROM EMP;
COUNT (*)
-------
4
```

ANALYSIS
It counts only distinct jobs

SUM

SUM does just that. It returns the sum of all values in a column.

INPUT:

```
SQL> SELECT SUM(SAL) TOTAL_SALARY  FROM EMP;
```

OUTPUT:

```
TOTAL_SALARY
-------------
29025
```

ANALYSIS

Find the total salary drawn by all the employees

INPUT/OUTPUT:

```
SQL> SELECT SUM(SAL) TOTAL_SALARY, SUM(COMM)
TOTAL_COMM,
FROM EMP;
TOTAL_SALARY   TOTAL_COMM
-------------  ----------
        29025        2200
```

INPUT/OUTPUT:

```
SQL> SQL> SELECT SUM(SAL) TOTAL_SALARY, SUM(COMM)
TOTAL_COMM,
FROM EMP WHERE JOB = 'SALESMAN';
TOTAL_SALARY   TOTAL_COMM
-------------  ----------
         5600        2200
```

AVG

The **AVG** function computes the average of a column.

INPUT:

```
SQL> SELECT AVG(sal) average_salary
2 FROM EMP;
```

OUTPUT:

```
AVERAGE_SALARY
---------------
2073.21429
```

ANALYSIS

Find the average salary of all the employees

INPUT:

```
SQL> SELECT AVG(COMM) average_comm FROM EMP;
```

OUTPUT:

```
AVERAGE_COMM
------------
         550
```

ANALYSIS

Functions ignores null rows

MAX

INPUT:

```
SQL> SELECT MAX(SAL) FROM EMP;
```

OUTPUT:

```
 MAX(SAL)
 --------
        5000
```

ANALYSIS

Takes the value from one different rows from one particular column

INPUT:

```
SQL> SELECT MAX(ENAME) FROM EMP;
```

OUTPUT:

```
 MAX(ENAME)
 --------
        WARD
```

ANALYSIS

Max of name is identified based on ASCII value

INPUT:

```
SQL> SELECT MAX (hiredate) FROM EMP;
```

OUTPUT:

```
 MAX(HIREDATE)
 -------------
       12-JAN-83
```

MIN

INPUT:

```
SQL> SELECT MIN(SAL) FROM EMP;
```

OUTPUT:

```
 MIN(SAL)
 --------
      800
```

INPUT:

```
SQL> SELECT MIN(ENAME) FROM EMP;
```

OUTPUT:

```
 MIN (ENAME)
 --------
      ADAMS
```

INPUT

```
SELECT SUM(SAL),AVG(SAL),MIN(SAL),MAX(SAL),COUNT(*) FROM
EMP;
```

OUTPUT

```
 SUM(SAL)  AVG(SAL)       MIN(SAL)  MAX(SAL)  COUNT(*)
 --------  ----------     --------  --------  --------
   29025   2073.21429          800      5000        14
```

ANALYSIS

All the aggregate functions can be used together in a single SQL statement

SINGLE ROW FUNCTIONS

These functions work on each and every row and return a value to the calling places.

These functions are classified into different types

- Arithmetic Functions
- Character Functions
- Date functions
- Miscellaneous Functions

Arithmetic Functions

Many of the uses you have for the data you retrieve involve mathematics. Most Implementations of SQL provide arithmetic functions similar to that of operators covered here.

ABS

The **ABS** function returns the absolute value of the number you point to. For example:

INPUT:

```
SQL> SELECT ABS(-10) ABSOLUTE_VALUE FROM dual;
```

OUTPUT

```
ABSOLUTE_VALUE
----------------------------
                          10
```

ANALYSIS

ABS changes all the negative numbers to positive and leaves positive numbers alone.

Dual is a system table or dummy table from where we can display system information (i.e. system date and username etc) or we can make our own calculations

CEIL and FLOOR

CEIL returns the smallest integer greater than or equal to its argument. **FLOOR** does just the reverse, returning the largest integer equal to or less than its argument.

INPUT:

```
SQL> SELECT CEIL(12.145) FROM DUAL;
```

OUTPUT:

```
CEIL(12.145)
-----------------
             13
```

INPUT:

```
SQL> SELECT CEIL(12.000) FROM DUAL;
```

OUTPUT:

```
CEIL(12.000)
-----------------
             12
```

ANALYSIS

Minimum we require one decimal place , to get the next higher integer number

INPUT/OUTPUT:

```
SQL> SELECT FLOOR(12.678) FLOOR DUAL;
```

OUTPUT:

```
FLOOR(12.678)
-----------------
             12
```

INPUT:

```
SQL> SELECT FLOOR(12.000) FROM DUAL;
```

OUTPUT:

```
FLOOR(12.000)
-----------------
             12
```

MOD

It returns remainder when we divide one value with another value

INPUT

```
SQL> SELECT MOD(5,2) FROM DUAL;
```

OUTPUT:

```
   MOD(5,2)
 ---------------
               1
```

INPUT

```
SQL> SELECT MOD(2,5) FROM DUAL;
```

OUTPUT:

```
   MOD(2,5)
 ---------------
               2
```

ANALYSIS

When numerator value less than denominator, it returns numerator value as remainder.

POWER

To raise one number to the power of another, use **POWER**. In this function the first argument is raised to the power of the second:

```
INPUT:
SQL> SELECT POWER(5,3) FROM DUAL;

OUTPUT:
    125
```

SIGN

SIGN returns **-1** if its argument is less than **0**, **0** if its argument is equal to **0**, and **1** if its argument is greater than **0**,

```
INPUT:
SQL> SELECT SIGN(-10), SIGN(10),SIGN(0) FROM DUAL;

OUTPUT:
SIGN(-10)   SIGN(10)    SIGN(0)
---------- ---------- ----------
        -1          1          0
```

SQRT

The function SQRT returns the square root of an argument. Because the square root of a negative number is undefined, you cannot use SQRT on negative numbers.

```
INPUT/OUTPUT:
SQL> SELECT SQRT(4) FROM DUAL;
       2
```

Character Functions

Many implementations of SQL provide functions to manipulate characters and strings of characters.

CHR

CHR returns the character equivalent of the number it uses as an argument. The character it returns depends on the character set of the database. For this example the database is set to ASCII.

INPUT:

```
SQL> SELECT CHR(65) FROM DUAL;
```

OUTPUT:

```
A
```

CONCAT

It is similar to that of concatenate operator (| |)

INPUT:

```
SQL> SELECT CONCAT(‘KRISHNA’, ‘ KANTH’) FROM DUAL;
```

OUTPUT

```
KRISHNA KANTH
```

INITCAP

INITCAP capitalizes the first letter of a word and makes all other characters lowercase.

INPUT:

```
SQL> SELECT ENAME “BEFORE”,INITCAP(ENAME) “AFTER”FROM EMP;
```

OUTPUT:

```
BEFORE     AFTER
---------- ----------
SMITH      Smith
ALLEN      Allen
WARD       Ward
JONES      Jones
MARTIN     Martin
BLAKE      Blake
CLARK      Clark
SCOTT      Scott
KING       King
TURNER     Turner
ADAMS      Adams
JAMES      James
FORD       Ford
MILLER     Miller
```

LOWER and UPPER

As you might expect, **LOWER** changes all the characters to lowercase; **UPPER** does just the changes all the characters to uppercase.

```
SQL>SELECT ENAME,UPPER(ENAME) UPPER_CASE,LOWER(ENAME)
LOWER_CASE FROM EMP;

ENAME      UPPER_CASE LOWER_CASE
---------- ---------- ----------
SMITH      SMITH      smith
ALLEN      ALLEN      allen
WARD       WARD       ward
JONES      JONES      jones
MARTIN     MARTIN     martin
BLAKE      BLAKE      blake
CLARK      CLARK      clark
SCOTT      SCOTT      scott
KING       KING       king
TURNER     TURNER     turner
ADAMS      ADAMS      adams
JAMES      JAMES      james
FORD       FORD       ford
MILLER     MILLER     miller
```

LPAD and RPAD

LPAD and **RPAD** take a minimum of two and a maximum of three arguments. The first argument is the character string to be operated on. The second is the number of characters to pad it with, and the optional third argument is the character to pad it with. The third argument defaults to a blank, or it can be a single character or a character string.

The following statement adds five pad characters, assuming that the field **LASTNAME** is defined as a 15-character field:

INPUT:

```
SQL> SELECT LPAD(ENAME,15,’*’) FROM EMP;
```

OUTPUT:

```
LPAD(ENAME,15,'
---------------
**********SMITH
**********ALLEN
***********WARD
**********JONES
*********MARTIN
**********BLAKE
**********CLARK
**********SCOTT
***********KING
*********TURNER
**********ADAMS
**********JAMES
***********FORD
*********MILLER
```

ANALYSIS:

15 locations allocated to display ename, out of that, name is occupying some space and in the remaining space to the left side of the name pads with *

INPUT

```
SQL> SELECT RPAD(5000,10,’*’) FROM DUAL;
```

OUTPUT:

5000****

LTRIM and RTRIM

LTRIM and RTRIM take at least one and at most two arguments. The first argument, like LPAD and RPAD, is a character string. The optional second element is either a character or character string or defaults to a blank. If you use a second argument that is not a blank, these trim functions will trim that character the same way they trim the blanks in the following examples.

INPUT:

```
SQL> SELECT ENAME, RTRIM(ENAME,'R') FROM EMP;
```

OUTPUT:

```
ENAME      RTRIM(ENAM
---------- ----------
SMITH      SMITH
ALLEN      ALLEN
WARD       WARD
JONES      JONES
MARTIN     MARTIN
BLAKE      BLAKE
CLARK      CLARK
SCOTT      SCOTT
KING       KING
TURNER     TURNE
ADAMS      ADAMS
JAMES      JAMES
FORD       FORD
MILLER     MILLE
```

ANALYSIS

Removes the rightmost character

REPLACE

REPLACE does just that. Of its three arguments, the first is the string to be searched. The second is the search key. The last is the optional replacement string. If the third argument is left out or **NULL**, each occurrence of the search key on the string to be searched is removed and is not replaced with anything.

SYNTAX :

REPLACE(STRING,SEARCH_STRING,REPLACE_STRING)

INPUT:

```
SQL> SELECT REPLACE (‘RAMANA’,’MA’, VI’) FROM DUAL;
```

OUTPUT

```
        RAVINA
```

INPUT

```
SQL> SELECT REPLACE(‘RAMANA’,’MA’) FROM DUAL;
```

OUTPUT

```
        RANA
```

ANALYSIS

When the replace string is missing, search string removed from the given string

INPUT

```
SQL> SELECT REPLACE (‘RAMANA’,’MA’, NULL) FROM DUAL;
```

OUTPUT

```
        RANA
```

TRANSLATE

The function **TRANSLATE** takes three arguments: the target string, the **FROM** string, and the **TO** string. Elements of the target string that occur in the **FROM** string are translated to the corresponding element in the **TO** string.

INPUT:

```
SQL> SELECT TRANSLATE(‘RAMANA’,’MA’,’CD’) FROM DUAL;
```

OUTPUT:

```
RDCDND
```

ANALYSIS

Notice that the function is case sensitive.

When search string matches, it replaces with corresponding replace string and if any one character is matching in the search string , it replaces with corresponding replace

SUBSTR

This three-argument function enables you to take a piece out of a target string. The first argument is the target string. The second argument is the position of the first character to be output. The third argument is the number of characters to show.

SYNTAX

SUBSTR(STRING,STARTING_POSITION[,NO_OF_CHARACTERS])

INPUT:

```
SQL> SELECT SUBSTR(‘RAMANA’,1,3) FROM DUAL;
```

OUTPUT:

```
RAM
```

ANALYSIS

It takes first 3 characters from first character

INPUT:

```
SQL> SELECT SUBSTR(‘RAMANA’,3,3) FROM DUAL;
```

OUTPUT:

```
     MAN
```

ANALYSIS

It takes 3 characters from third position

INPUT:

```
SQL> SELECT SUBSTR(‘RAMANA’,-2,2) FROM DUAL;
```

OUTPUT:

```
     NA
```

ANALYSIS

You use a negative number as the second argument, the starting point is determined by counting backwards from the end.

INPUT:

```
SQL> SELECT SUBSTR(‘RAMANA’,1,2) || SUBSTR(‘RAMANA’,-2,2)
FROM DUAL;
```

OUTPUT:

```
     RANA
```

ANALYSIS

First two characters and last two characters are combined together as a single string

INPUT:

```
SQL> SELECT SUBSTR(‘RAMANA’,3) FROM DUAL;
```

OUTPUT:

```
     MANA
```

ANALYSIS

When third argument is missing, it takes all the character from starting position

INPUT:

```
SQL> SELECT * FROM EMP WHERE SUBSTR(HIREDATE,4,3) =
SUBSTR(SYSDATE,4,3);
```

OUTPUT:

```
    RANA
```

ANALYSIS

Displays all the employees who joined in the current month
SYSDATE is a single row function, which gives the current date.

INPUT:

```
SQL> SELECT SUBSTR(‘RAMANA’,1,2) || SUBSTR(‘RAMANA’,-2,2)
FROM DUAL;
```

OUTPUT:

```
    RANA
```

ANALYSIS

First two characters and Last two characters are combined together as a single string

INSTR

To find out where in a string a particular pattern occurs, use **INSTR**. Its first argument is the target string. The second argument is the pattern to match. The third and forth are numbers representing where to start looking and which match to report.

This example returns a number representing the first occurrence of **O** starting with the second

INPUT

```
SQL> SELECT INSTR(‘RAMANA’,’A’) FROM DUAL;
```

OUTPUT

```
      2
```

ANALYSIS

Find the position of the first occurrence of letter A

INPUT

```
SQL> SELECT INSTR(‘RAMANA’,’A’,1,2) FROM DUAL;
```

OUTPUT

```
SQL> SELECT INSTR (‘RAMANA’,’a’) FROM DUAL;

 OUTPUT
            0
```

ANALYSIS

Function is case sensitive; it returns 0 (zero) when the given character is not found.

INPUT

```
SQL> SELECT INSTR(‘RAMANA’,’A’,3,2) FROM DUAL;

 OUTPUT
            6
```

ANALYSIS

Find the position of the second occurrence of letter A from 3rd position of the string

Conversion Functions

These functions provide a handy way of converting one type of data to another. They are mainly useful for changing date formats and number formats.

TO_CHAR

The primary use of **TO_CHAR** is to convert a number into a character. Different Implementations may also use it to convert other data types, like Date, into a character, or to include different formatting arguments.

The following example illustrates the primary use of **TO_CHAR**:

INPUT:

```
SQL> SELECT SAL, TO_CHAR(SAL) FROM EMP;
```

OUTPUT:

```
       SAL TO_CHAR(SAL)
---------- ----------------------------------------
       800 800
      1600 1600
      1250 1250
      2975 2975
      1250 1250
      2850 2850
      2450 2450
      3000 3000
      5000 5000
      1500 1500
      1100 1100
       950 950
      3000 3000
      1300 1300
```

ANALYSIS

After conversion, Converted information is left aligned. So we can say that it is a string.

The main usage of this function is, to change the date formats and number formats

INPUT:

```
SQL> SELECT SYSDATE,TO_CHAR(SYSDATE,’DD/MM/YYYY’) FROM
DUAL;
```

OUTPUT:

```
SYSDATE     TO_CHAR(SYSDATE,'DD/MM/YYYY')
---------   ------------------------------
24-MAR-07   24/03/2007
```

ANALYSIS

Convert the default date format to DD/MM/YYYY format

INPUT:

```
SQL> SELECT SYSDATE,TO_CHAR(SYSDATE,’DD-MON-YY’) FROM
DUAL;
```

OUTPUT:

```
SYSDATE     TO_CHAR(SYSDATE,'DD-MON-YY')
---------   ------------------------------
24-MAR-07   24-MAR-07
```

INPUT:

```
SQL> SELECT SYSDATE,TO_CHAR(SYSDATE,’DY-MON-YY’) FROM
DUAL;
```

OUTPUT:

```
SYSDATE     TO_CHAR(SYSDATE,'DY-MON-YY')
---------   ------------------------------
24-MAR-07   SAT-MAR-07
```

ANALYSIS:

DY displays the first 3 letters from the day name

INPUT:

```
SQL> SELECT SYSDATE,TO_CHAR(SYSDATE,'DAY MONTH YEAR') FROM
DUAL;
```

OUTPUT:

```
SYSDATE      TO_CHAR(SYSDATE,'DAYMONTHYEAR')
---------     ------------------------------
24-MAR-07     SATURDAY  MARCH TWO THOUSAND SEVEN
```

ANALYSIS:

DAY	gives the total day name
MONTH	gives the total month name
YEAR	writes the year number in words

INPUT:

```
SQL> SELECT SYSDATE,TO_CHAR(SYSDATE,'DDSPTH MONTH YEAR')
FROM DUAL;
```

OUTPUT:

```
SYSDATE   TO_CHAR(SYSDATE,'DDSPTHMONTHYEAR')
--------- ------------------------------------------------------------------
24-MAR-07 TWENTY-FOURTH MARCH     TWO THOUSAND SEVEN
```

ANALYSIS:

DD	gives the day number
DDSP	Writes day number in words
TH	is the format. Depends upon the number it gives either ST / RD/ST/ND format

INPUT:

```
SQL> SELECT HIREDATE,TO_CHAR(HIREDATE,’DDSPTH MONTH
YEAR’) FROM EMP;
```

OUTPUT:

```
HIREDATE  TO_CHAR(HIREDATE,'DDSPTHMONTHYEAR')
--------- ------------------------------------------------------------------
17-DEC-80 SEVENTEENTH DECEMBER  NINETEEN EIGHTY
20-FEB-81 TWENTIETH FEBRUARY  NINETEEN EIGHTY-ONE
22-FEB-81 TWENTY-SECOND FEBRUARY  NINETEEN EIGHTY-ONE
02-APR-81 SECOND APRIL     NINETEEN EIGHTY-ONE
28-SEP-81 TWENTY-EIGHTH SEPTEMBER NINETEEN EIGHTY-ONE
01-MAY-81 FIRST MAY       NINETEEN EIGHTY-ONE
09-JUN-81 NINTH JUNE      NINETEEN EIGHTY-ONE
09-DEC-82 NINTH DECEMBER  NINETEEN EIGHTY-TWO
17-NOV-81 SEVENTEENTH NOVEMBER  NINETEEN EIGHTY-ONE
08-SEP-81 EIGHTH SEPTEMBER NINETEEN EIGHTY-ONE
12-JAN-83 TWELFTH JANUARY   NINETEEN EIGHTY-THREE
03-DEC-81 THIRD DECEMBER  NINETEEN EIGHTY-ONE
03-DEC-81 THIRD DECEMBER  NINETEEN EIGHTY-ONE
23-JAN-82 TWENTY-THIRD JANUARY   NINETEEN EIGHTY-TWO
```

ANALYSIS:
Converts all hire dates in EMP table into Words

INPUT:

```
SQL> SELECT SYSDATE,TO_CHAR(SYSDATE,’Q’) FROM DUAL;
```

OUTPUT:

```
SYSDATE      TO_CHAR(SYSDATE,’Q’)
---------    ------------------------------
24-MAR-07    1
```

ANALYSIS:

Gives in the quarter the given date falls

INPUT:

```
SQL> SELECT TO_CHAR(TO_DATE(’10-SEP-2005’),’Q’) FROM DUAL;
```

OUTPUT:

```
TO_CHAR(TO_DATE('10-SEP-2005'),'Q')
---------------------------------------
3
```

ANALYSIS:

To_date is data conversion function, which converts given string into date type

INPUT:

```
SQL> SELECT SYSDATE,TO_CHAR(SYSDATE,’W’) FROM DUAL;
```

OUTPUT:

```
SYSDATE     TO_CHAR(SYSDATE,’W’)
---------    ------------------------------
24-MAR-07    4
```

ANALYSIS:

Gives the week number in the current month (In which week given date falls in the current month)

INPUT:

```
SQL> SELECT SYSDATE,TO_CHAR(SYSDATE,’WW’) FROM DUAL;
```

OUTPUT:

```
SYSDATE    TO_CHAR(SYSDATE)
---------    ----------------------------
24-MAR-07   12
```

ANALYSIS:

Returns no. of weeks worked during the year.

INPUT:

```
SQL> SELECT TO_CHAR(SYSDATE,’HH:MI:SS AM’) FROM DUAL;
```

OUTPUT:

```
TO_CHAR(SYS
-----------
08:40:17 PM
```

ANALYSIS:

HH   returns Hours   }
MI   returns Minutes   } Returns time from current date
SS   returns Seconds   }
AM returns AM / PM depends on Time

INPUT:

```
SQL> SELECT TO_CHAR(SYSDATE,’HH24:MI:SS’) FROM DUAL;
```

OUTPUT:

```
TO_CHAR(
--------
20:43:12
```

ANALYSIS:

HH24   returns Hours in 24 hour format   }
MI   returns Minutes   } Returns time from current date
SS   returns Seconds   }

INPUT:

```
SQL> SELECT TO_CHAR(12567,’99,999.99’) FROM DUAL;
```

OUTPUT:

```
TO_CHAR(12567,'99,999.99')
-----------------------------
 12,567.00
```

ANALYSIS:

Converts the given number into comma format with two decimal places

INPUT:

```
SQL> SELECT TO_CHAR(12567,’L99,999.99’) FROM DUAL;
```

OUTPUT:

```
TO_CHAR(12567,'L99,999.99')
------------------------------
 $12,567.00
```

ANALYSIS:

Display the local currency symbol

INPUT:

```
SQL> SELECT TO_CHAR(-12567,’L99,999.99PR’) FROM DUAL;
```

OUTPUT:

```
TO_CHAR(-12567,'L99,999.99PR')
------------------------------------
        <$12,567.00>
```

ANALYSIS:

PR   Parenthesis negative number

TO_NUMBER

TO_NUMBER is the companion function to TO_CHAR, and of course, it converts a string into a number.

For example:

INPUT:

```
SQL> SELECT SAL, TO_NUMBER((TO_CHAR(SAL)) FROM EMP;
```

OUTPUT:

```
       SAL TO_NUMBER(TO_CHAR(SAL))
---------- -----------------------
       800                     800
      1600                    1600
      1250                    1250
      2975                    2975
      1250                    1250
      2850                    2850
      2450                    2450
      3000                    3000
      5000                    5000
      1500                    1500
      1100                    1100
       950                     950
      3000                    3000
      1300                    1300
```

ANALYSIS

After conversion, Converted information is right aligned. So we can say that it is a number.

Date and Time Functions

We live in a civilization governed by times and dates, and most major implementations of SQL have functions to cope with these concepts.

It demonstrates the time and date functions.

ADD_MONTHS

This function adds a number of months to a specified date.

For example, say a customer deposited some amount on a particular date for a period of 6 months. To find the maturity date of the deposit

INPUT:

```
SQL> SELECT ADD_MONTHS (SYSDATE, 6) MATURITY_DATE FROM
DUAL;
```

OUTPUT:

```
MATURITY_DATE
--------------------
24-SEP-07
```

ANALYSIS

It adds 6 months to the system date

INPUT:

```
SQL> SELECT HIREDATE,
TO_CHAR(ADD_MONTHS(HIREDATE,33*12),’DD/MM/YYYY’)
RETIRE_DATE  FROM EMP;
```

OUTPUT:

```
HIREDATE  RETIRE_DATE
--------- ---------------
17-DEC-80 17/12/2013
20-FEB-81 20/02/2014
22-FEB-81 22/02/2014
02-APR-81 02/04/2014
28-SEP-81 28/09/2014
01-MAY-81 01/05/2014
09-JUN-81 09/06/2014
09-DEC-82 09/12/2015
17-NOV-81 17/11/2014
08-SEP-81 08/09/2014
12-JAN-83 12/01/2016
03-DEC-81 03/12/2014
03-DEC-81 03/12/2014
23-JAN-82 23/01/2015
```

ANALYSIS

Displaying the retirement date with century

INPUT:

```
SQL> SELECT HIREDATE, ADD_MONTHS(HIREDATE,33*12)
RETIRE_DATE  FROM EMP;
```

OUTPUT:

```
HIREDATE  RETIRE_DATE
--------- ---------------
17-DEC-80 17-DEC-13
20-FEB-81 20-FEB-14
22-FEB-81 22-FEB-14
02-APR-81 02-APR-14
28-SEP-81 28-SEP-14
01-MAY-81 01-MAY-14
09-JUN-81 09-JUN-14
09-DEC-82 09-DEC-15
17-NOV-81 17-NOV-14
08-SEP-81 08-SEP-14
12-JAN-83 12-JAN-16
03-DEC-81 03-DEC-14
03-DEC-81 03-DEC-14
23-JAN-82 23-JAN-15
```

ANALYSIS

Find the retirement date of an employee

Assume, 33 years of service from date of join is retirement date

LAST_DAY

LAST_DAY returns the last day of a specified month.
For example, you need to know what the last day of the month

MONTHS_BETWEEN

Used to find the number of months between two given months

INPUT:

```
SQL> SELECT LAST_DAY(SYSDATE) FROM DUAL;
```

OUTPUT:

```
LAST_DAY(SYSDATE)
-------------------------
31-MAR-07
```

ANALYSIS

Find the last date of the month

INPUT:

```
SQL> SELECT ENAME,MONTHS_BETWEEN(SYSDATE,HIREDATE)/12
   EXPERIENCE FROM EMP;
```

OUTPUT:

```
ENAME      EXPERIENCE
---------- ----------
SMITH      26.2713494
ALLEN      26.0966182
WARD       26.0912419
JONES      25.9783387
MARTIN     25.4917795
BLAKE      25.8976935
CLARK      25.7928548
SCOTT      24.2928548
KING       25.3546827
TURNER      25.545543
ADAMS      24.2014569
JAMES      25.3089838
FORD       25.3089838
MILLER      25.171887
```

ANALYSIS

Finds number of months between sysdate and hiredate . Result is divided with 12 to get the experience

LOCALTIMESTAMP

Returns Local timestamp in the active time zone, with no time zone information shown

INPUT

```
SQL> SELECT LOCALTIMESTAMP FROM DUAL;
```

OUTPUT

```
LOCALTIMESTAMP
------------------------------------
25-MAR-07 06.21.02.312000 PM
```

NEW_TIME

This function is used to adjust the time according to the time zone you are in.

Here are the time zones you can use with this function:

Abbreviation Time Zone
AST or ADT Atlantic standard or daylight time
BST or BDT Bering standard or daylight time
CST or CDT Central standard or daylight time
EST or EDT Eastern standard or daylight time
GMT Greenwich mean time
HST or HDT Alaska-Hawaii standard or daylight time
MST or MDT Mountain standard or daylight time
NST Newfoundland standard time
PST or PDT Pacific standard or daylight time
YST or YDT Yukon standard or daylight time
You can adjust your time like this:

```
INPUT
SQL> select
TO_CHAR(new_time(LOCALTIMESTAMP,'EsT','PDT'),'DD/MM/YYYY HH : MI
:SS PM') from DUAL;

OUTPUT
TO_CHAR(NEW_TIME(LOCALTIM
-------------------------
25/03/2007 04 : 32 :55 PM
```

INPUT:

```
SQL> select hiredate,new_time(hiredate,'EsT','PDT') from
emp;
```

OUTPUT:

```
HIREDATE  NEW_TIME(
--------- ---------
17-DEC-80 16-DEC-80
20-FEB-81 19-FEB-81
22-FEB-81 21-FEB-81
02-APR-81 01-APR-81
28-SEP-81 27-SEP-81
01-MAY-81 30-APR-81
09-JUN-81 08-JUN-81
09-DEC-82 08-DEC-82
17-NOV-81 16-NOV-81
08-SEP-81 07-SEP-81
12-JAN-83 11-JAN-83
03-DEC-81 02-DEC-81
03-DEC-81 02-DEC-81
23-JAN-82 22-JAN-82
```

ANALYSIS

Like magic, all the times are in the new time zone and the dates are adjusted

NEXT_DAY

NEXT_DAY finds the name of the first day of the week that is equal to or later than another specified date.

INPUT:

```
SQL> SELECT NEXT_DAY(SYSDATE,’MONDAY’) FROM DUAL;
```

OUTPUT:

```
NEXT_DAY(
---------
26-MAR-07
```

ANALYSIS
If the sysdate is Saturday, March 24, 2007, It display the date of the next coming Monday.

EXTRACT

We can use Extract function in the place of to_char function from Oracle 9i.

```
SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL;

SELECT EXTRACT(DAY FROM SYSDATE) FROM DUAL

SELECT EXTRACT(YEAR FROM SYSDATE) FROM DUAL
```

Interval Data types (Only from 9i)

Oracle supports two interval datatypes. Both were introduced in Oracle9i Database, and both conform to the ISO SQL standard.

INTERVAL YEAR to MONTH
Allows you to define and interval of time in terms of years and months

INTERVAL DAY TO SECOND
Allows you to define an interval of time in terms of days, hours, minutes and seconds (including fractional seconds).

Example

```
select (sysdate - to_date('10-jan-2004')) year to  month from dual
```

OUTPUT

```
+000000003-10
```

Analysis

Returns the difference. No of years and months between both the dates.

Example

```
Select (sysdate - to_date('10-NOV-2007')) day to second from dual
```

OUTPUT

```
+000000009 11:04:07
```

Analysis

Find the difference with days and time. For sysdate it takes 0.00 hrs as starting time

Note :- We can also change the default format for a given session using the ALTER SESSION command

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YYYY';


SQL> SELECT SYSDATE FROM DUAL:
```

Interpreting Two-digit Years

Oracle provides the RR format element to interpret two-digit years.

If the Current year is in the first half of the century (years 0 through 49), then:

- If you enter a date in the first half of the century (i.e. from 0 through 49), RR returns the current century.
- If you enter a date in the latter half of the century (i.e. from 50 through 99), RR returns the previous century.

For example

```
select to_char(sysdate,'dd/mm/yyyy') "current date",
to_char(to_date('14-oct-88','dd-mon-rr'),'yyyy') "Year 88",
to_char(to_date('14-oct-18','dd-mon-rr'),'yyyy') "year 18" from dual

OUTPUT

current date          Year 88          year 18
------------------------- -------------------- --------------------
19/11/2007              1988             2018
```

When we reach the year 2050, RR will interpret the same dates differently

```
current date          Year 88          year 18
------------------------- -------------------- --------------------
19/11/2050              2088             2118
```

Going from Numbers to Intervals

The NUMTOYMINTERVAL and NUMTODSINTERVAL functions allow you to convert a single numeric value to one of the interval data types.

The function NUMTOYMINTERVAL (pronounced “ num to Y M interval”) converts a numeric value an interval of type INTERVAL YEAR TO MONTH.

```
SQL> select numtoyminterval (10.5,'year') from dual;


NUMTOYMINTERVAL (10.5,'YEAR')
---------------------------------------------------------------------------
+000000010-06

SQL> select numtoyminterval (10.3,'year') from dual;

NUMTOYMINTERVAL (10.3,'YEAR')
---------------------------------------------------------------------------
+000000010-03
```

Try the following formats

- SELECT NUMTOYMINTERVAL(10.5,’MONTH’) FROM DUAL;
- SELECT NUMTOYMINTERVAL(10.5,’DAY’) FROM DUAL;

Name	**Description**
YEAR	Some number of years ranging from 1 through 999,999,999
MONTH	Some number of months ranging from 0 through 11
DAY	Some number of days ranging from 0 through 999,999,999
HOUR	Some number of hours ranging from 0 through 23
MINUTE	Some number of minutes ranging from 0 through 59
SECOND	Some number of seconds ranging from 0 through 59.999999999

NUMTODSINTERVAL

This function allows to convert a single numeric value to one of the interval data types.

This function (pronounced “num to D S interval”) likewise converts a numeric value to an interval of type INTERVAL DAY TO SECOND.

Example

```
SQL> select NUMTODSINTERVAL(1440,’minute’) from dual;
```

OUTPUT

```
+01 00:00.00.000000
```

ANALYSIS
Oracle automatically taken care of normalizing the input value of 1440 minutes to an interval value of 1 day.

- **Miscellaneous Functions**

Here are three miscellaneous functions you may find useful.
GREATEST and LEAST

INPUT:

```
SQL> SELECT GREATEST(10,1,83,2,9,67) FROM DUAL;
```

OUTPUT:

```
GREATEST
---------
  83
```

ANALYSIS

Displays the greatest of the given set of values

Difference between GREATEST AND MAX IS

1) GREATEST IS SINGLE ROW FUNCTION, MAX IS A GROUP FUNCTION
2) GREATEST TAKES VALUES FROM DIFFERENT COLUMNS FROM EACH ROW, WHERE AS MAX TAKES VALUES FROM DIFFERENT ROWS FROM A COLUMN.

Assume there is a student table

STUDENT

ROLLNO	NAME	SUB1	SUB2	SUB3	SUB4
1	RAVI	55	22	86	45
2	KRIS	78	55	65	12
3	BABU	55	22	44	77
4	ANU	44	55	66	88

To find the greatest and Least Marks

INPUT:

```
SQL> SELECT NAME,SUB1,SUB2,SUB3, SUB4,
GREATEST(SUB1,SUB2,SUB3,SUB4) GREATEST_MARK,
LEAST(SUB1,SUB2,SUB3,SUB4) LEAST_MARK FROM STUDENT
```

OUTPUT:

ROLLNO	**NAME**	**SUB1**	**SUB2**	**SUB3**	**SUB4**	**GREATEST MARK**	**LEAST MARK**
1	**RAVI**	**55**	**22**	**86**	**45**	**86**	**22**
2	**KRIS**	**78**	**55**	**65**	**12**	**78**	**12**
3	**BABU**	**55**	**22**	**44**	**77**	**77**	**22**
4	**ANU**	**44**	**55**	**66**	**88**	**88**	**44**

USER

USER returns the character name of the current user of the database.

INPUT:

```
SQL> SELECT USER FROM DUAL;
```

OUTPUT:

```
USER
--------
SCOTT
```

ANALYSIS

Displays the current sessions user name

We can also display username using environment command

SQL> SHOW USER

The DECODE Function

The DECODE function is one of the most powerful commands in SQL*Plus--and perhaps the most powerful. The standard language of SQL lacks procedural functions that are contained in languages such as COBOL and C.
The DECODE statement is similar to an IF...THEN statement in a procedural programming language. Where flexibility is required for complex reporting needs, DECODE is often able to fill the gap between SQL and the functions of a procedural language.

SYNTAX:

```
DECODE (column1, value1, output1, value2, output2, output3)
```

The syntax example performs the DECODE function on column1.

If column1 has a value of value1, then display output1 instead of the column's current value.

If column1 has a value of value2, then display output2 instead of the column's current value.

If column1 has a value of anything other than value1 or value2, then display output3 instead of the column's current value.

INPUT

SQL> SELECT ENAME,JOB,DECODE(JOB,’CLERK’,’EXEC’,’SALESMAN’,
‘S.OFFICER’,’ANALYST’,’PM’,’MANAGER’,’VP’,JOB) PROMOTION FROM EMP;

OUTPUT

```
ENAME      JOB       PROMOTION
---------- --------- ---------
SMITH      CLERK     EXEC
ALLEN      SALESMAN  S.OFFICER
WARD       SALESMAN  S.OFFICER
JONES      MANAGER   VP
MARTIN     SALESMAN  S.OFFICER
BLAKE      MANAGER   VP
CLARK      MANAGER   VP
SCOTT      ANALYST   PM
KING       PRESIDENT PRESIDENT
TURNER     SALESMAN  S.OFFICER
ADAMS      CLERK     EXEC
JAMES      CLERK     EXEC
FORD       ANALYST   PM
MILLER     CLERK     EXEC
```

ANALYSIS

When JOB has a value CLERK , then display EXEC instead of CLERK
When JOB has a value SALESMAN , then display S.OFFICER instead of SALESMAN
When JOB has a value ANALYST , then display PM instead of ANALYST
When JOB has a value MANAGER , then display VP instead of MANAGER
OTHERWISE DISPLAY SAME JOB

INPUT

SQL> **SELECT ENAME,JOB,SAL,DECODE(JOB,’CLERK’,SAL*1.1,’SALESMAN’, SAL*1.2,’ANALYST’,SAL*1.25,’MANAGER’,SAL*1.3,SAL) NEW_SAL FROM EMP;**

OUTPUT

```
ENAME      JOB              SAL    NEW_SAL
---------- --------- ---------- ----------
SMITH      CLERK            800        880
ALLEN      SALESMAN        1600       1920
WARD       SALESMAN        1250       1500
JONES      MANAGER         2975     3867.5
MARTIN     SALESMAN        1250       1500
BLAKE      MANAGER         2850       3705
CLARK      MANAGER         2450       3185
SCOTT      ANALYST         3000       3750
KING       PRESIDENT       5000       5000
TURNER     SALESMAN        1500       1800
ADAMS      CLERK           1100       1210
JAMES      CLERK            950       1045
FORD       ANALYST         3000       3750
MILLER     CLERK           1300       1430
```

ANALYSIS

When JOB has a value CLERK , then giving 10% increment
When JOB has a value SALESMAN , then giving 20% increment
When JOB has a value ANALYST , then giving 25% increment
When JOB has a value MANAGER , then giving 30% increment
OTHERWISE no increment

Assume there is a table with empno,ename,sex

INPUT

SQL> SELECT ENAME,SEX,DECODE(SEX,’MALE’,’MR.’||ENAME,
 ‘MS.’||ENAME) FROM EMP;

ANALYSIS

Adding Mr.’ or ‘Ms.’ before the name based on their Gender

CASE

As of Oracle 9i, you can use the CASE function in place of DECODE. The CASE function uses the keywords when, then, else, and end to indicate the logic path followed, which may make the resulting code easier to follow than an equivalent DECODE.

Example

```
SQL> SELECT JOB,
    CASE JOB
    WHEN 'MANAGER' then 'VP'
    WHEN 'CLERK'   THEN 'EXEC'
    WHEN 'SALESMAN' THEN 'S.OFFICER'
    ELSE
       JOB
    END
    FROM EMP;

JOB       CASEJOBWH
--------- ---------
CLERK     EXEC
SALESMAN  S.OFFICER
SALESMAN  S.OFFICER
MANAGER   VP
SALESMAN  S.OFFICER
MANAGER   VP
MANAGER   VP
ANALYST   ANALYST
PRESIDENT PRESIDENT
SALESMAN  S.OFFICER
CLERK     EXEC
CLERK     EXEC
ANALYST   ANALYST
CLERK     EXEC
```

ANALYSIS
Works similar to that of DECODE

NVL

If the value is NULL, this function is equal to substitute. If the value is not NULL, this function is equal to value. Substitute can be a literal number, another column, or a computation.

NVL is not restricted to numbers, it can be used with CHAR, VARCHAR2, DATE, and other data types, but the value and substitute must be the same data type.

SYNTAX NVL(value, substitute)

INPUT

SQL> SELECT EMPNO,SAL,COMM, SAL + COMM TOTAL FROM EMP;

OUTPUT

```
     EMPNO        SAL       COMM      TOTAL
---------- ---------- ---------- ----------
      7369        800
      7499       1600        300       1900
      7521       1250        500       1750
      7566       2975
      7654       1250       1400       2650
      7698       2850
      7782       2450
      7788       3000
      7839       5000
      7844       1500          0       1500
      7876       1100
      7900        950
      7902       3000
      7934       1300
```

ANALYSIS

Arithmetic operation is possible only when value is there in both columns

INPUT

SQL> SELECT EMPNO,SAL,COMM, SAL + NVL(COMM,0) TOTAL FROM EMP;

OUTPUT

```
     EMPNO        SAL       COMM      TOTAL
---------- ---------- ---------- ----------
      7369        800                   800
      7499       1600        300       1900
      7521       1250        500       1750
      7566       2975                  2975
      7654       1250       1400       2650
      7698       2850                  2850
      7782       2450                  2450
      7788       3000                  3000
      7839       5000                  5000
      7844       1500          0       1500
      7876       1100                  1100
      7900        950                   950
      7902       3000                  3000
      7934       1300                  1300
```

ANALYSIS

Using NVL, we are substituting 0 if COMM is NULL.

INPUT

SQL>SELECT DEPTNO,SUM(SAL),RATIO_TO_REPORT(SUM(SAL)) OVER() FROM EMP GROUP BY DEPTNO;

OUTPUT

```
    DEPTNO   SUM(SAL) RATIO_TO_REPORT(SUM(SAL))OVER()
---------- ---------- -------------------------------
        10       8750                      .301464255
        20      10875                      .374677003
        30       9400                      .323858742
```

ANALYSIS

RATIO_TO_REPORT FUNCTION FINDS THE SALARY RATIO OF THAT DEPARTMENT OVER THE TOTAL SALARY OF ALL THE EMPLOYEES.

LENGTH

Finds the length of the given information

```
SQL> SELECT ENAME,LENGTH(ENAME) FROM EMP;
SQL> SELECT LENGTH(SYSDATE) FROM EMP;
SQL> SELECT SAL,LENGTH(SAL) FROM EMP;
```

ASCII

Finds the ASCII value of the given character

```
SQL> SELECT ASCII('A') FROM DUAL;
```

CAST

Converts one type of information into another type

```
SQL> SELECT 50 numb, cast(50 as varchar2(2) value from dual;
```

Exercise

- _____________ function performs one to one character substitution.
- _____________ format option is used to get complete year spelled out in TO_CHAR function.
- ___________ symbol is used to combine tow given strings
- What happens if "replace string" is not given for REPLACE function
- Can a number be converted to DATE?
- Convert the value of name in the EMP table to lower case letters
- Display the names of the employees who have more than 4 characters in the name.
- Print *'s as number of thousands are there in the number
- Display the ename, comm. If the commission is NULL, print as NO COMM
- Add number of days to the given date
- Display the first and last two characters from a given name and combine them as a single string (Use only functions)
- Find the difference between two given dates
- Display all the names which contain underscore
- subtract number of months from given date

GROUP BY CLAUSE

GROUP BY CLAUSE

Group by statement groups all the rows with the same column value.
Use to generate summary output from the available data.
Whenever we use a group function in the SQL statement, we have to use a group by clause.

INPUT

```
SQL> SELECT JOB, COUNT (*) FROM EMP GROUP BY JOB;
```

OUTPUT

```
JOB         COUNT(*)
--------- ----------
ANALYST            2
CLERK              4
MANAGER            3
PRESIDENT          1
SALESMAN           4
```

ANALYSIS
Counts number of employees under each and every job.
When we are grouping on job, initially jobs are placed in ascending order in a temporary segment.
On the temporary segment, group by clause is applied, so that on each similar job count function applied.

INPUT

```
SQL> SELECT JOB, SUM (SAL) FROM EMP GROUP BY JOB;
```

OUTPUT

```
JOB         SUM(SAL)
--------- ----------
ANALYST         6000
CLERK           4150
MANAGER         8275
PRESIDENT       5000
SALESMAN        5600
```

ANALYSIS

With each job, it finds the total salary

ERROR with GROUP BY Clause

Note :

- Only grouped columns allowed in the group by clause
- When ever we are using a group function in the SQL statement, we have to use group by clause.

INPUT

SQL> SELECT JOB,COUNT(*) FROM EMP;

OUTPUT

```
SELECT JOB, COUNT (*) FROM EMP
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

ANALYSIS
This result occurs because the group functions, such as SUM and COUNT, are designated to tell you something about a group or rows, not the individual rows of the table. This error is avoided by using JOB in the group by clause, which forces the COUNT to count all the rows grouped within each job.

INPUT

```
SQL> SELECT JOB,ENAME,COUNT(*) FROM EMP GROUP BY JOB;
```

OUTPUT

```
SELECT JOB,ENAME,COUNT(*) FROM EMP GROUP BY JOB
           *
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

ANALYSIS

In the above query, JOB is only the grouped column where as ENAME column is not a grouped column.

What ever the columns we are grouping, the same column is allowed to display.

INPUT
SQL> SELECT JOB, MIN(SAL),MAX(SAL) FROM EMP GROUP BY JOB;

OUTPUT

```
JOB         MIN(SAL)   MAX(SAL)
--------- ---------- ----------
ANALYST         3000       3000
CLERK            800       1300
MANAGER         2450       2975
PRESIDENT       5000       5000
SALESMAN        1250       1600
```

ANALYSIS
With each job, it finds the MINIMUM AND MAXIMUM SALARY

For displaying Total summary information from the table.

INPUT
SQL> SELECT JOB, SUM(SAL),AVG(SAL),MIN(SAL),MAX(SAL) ,COUNT(*) FROM EMP GROUP BY JOB;

OUTPUT

```
JOB         SUM(SAL)   AVG(SAL)   MIN(SAL)   MAX(SAL)   COUNT(*)
--------- ---------- ---------- ---------- ---------- ----------
ANALYST         6000       3000       3000       3000          2
CLERK           4150     1037.5        800       1300          4
MANAGER         8275 2758.33333       2450       2975          3
PRESIDENT       5000       5000       5000       5000          1
SALESMAN        5600       1400       1250       1600          4
```

ANALYSIS

With each job, finds the total summary information.

To display the output Designation wise, Department wise total salaries
With a matrix style report.

INPUT

```
SQL> SELECT JOB,SUM(DECODE(DEPTNO,10,SAL)) DEPT10,
SUM(DECODE(DEPTNO,20,SAL)) DEPT20,
SUM(DECODE(DEPTNO,30,SAL)) DEPT30,
SUM(SAL) TOTAL FROM EMP GROUP BY JOB;
```

OUTPUT

```
JOB           DEPT10     DEPT20     DEPT30      TOTAL
--------- ---------- ---------- ---------- ----------
ANALYST                    6000                  6000
CLERK           1300       1900        950       4150
MANAGER         2450       2975       2850       8275
PRESIDENT       5000                             5000
SALESMAN                              5600       5600
```

ANALYSIS

When we apply group by, initially all the designations are placed in ascending order of designations.

Then group by clause groups similar designations, then DECODE function (Single row function) applies on each and every row of that group and checks the DEPTNO. If DEPTNO=10, it passes corresponding salary as an argument to SUM() .

INPUT

SQL> SELECT DEPTNO,JOB,COUNT(*) FROM EMP GROUP BY DEPTNO,JOB;

OUTPUT

```
    DEPTNO JOB         COUNT(*)
---------- --------- ----------
        10 CLERK              1
        10 MANAGER            1
        10 PRESIDENT          1
        20 CLERK              2
        20 ANALYST            2
        20 MANAGER            1
        30 CLERK              1
        30 MANAGER            1
        30 SALESMAN           4
```

ANALYSIS

To display the DEPTNO only one time

INPUT

SQL> BREAK ON DEPTNO SKIP 1
SQL> SELECT DEPTNO,JOB,COUNT(*) FROM EMP GROUP BY DEPTNO,JOB;

OUTPUT

```
    DEPTNO JOB         COUNT(*)
---------- --------- ----------
        10 CLERK              1
           MANAGER            1
           PRESIDENT          1

        20 CLERK              2
           ANALYST            2
           MANAGER            1

        30 CLERK              1
           MANAGER            1
           SALESMAN           4
```

ANALYSIS

Break is Environment command , which breaks the information on repetitive column and displays them only once.
SKIP 1 used with BREAK to leave one blank line after completion of each Deptno.

To remove the given break , we have to use an Environment command

SQL> CLEAR BREAK;

Group by with ROLLUP and CUBE Operators

- Use Rollup or CUBE with Group by to produce super aggregate rows by cross-referencing columns.
- ROLLUP grouping produces a result set containing the regular grouped rows and the subtotal values.
- CUBE grouping produces a result set containing the rows from ROLLUP and cross-tabulation rows

The ROLLUP and CUBE operators are available only in Oracle8i and later releases.

```
Syntax

SELECT [column,] group_function(column)...
FROM table
[WHERE condition]
[GROUP BY [CUBE] group_by_expression]
[HAVING having_expression]
[ORDER BY column];
```

CUBE function

We can use CUBE function to generate subtotals for all combinations of the values in the group by clause.(CUBE and ROLLUP are available only from 9i)

INPUT

SQL> SELECT DEPTNO,JOB,COUNT(*) FROM EMP GROUP BY CUBE(DEPTNO,JOB);

OUTPUT

```
    DEPTNO JOB         COUNT(*)
---------- --------- ----------
                             14
           CLERK              4
           ANALYST            2
           MANAGER            3
           SALESMAN           4
           PRESIDENT          1
        10                    3
        10 CLERK              1
        10 MANAGER            1
        10 PRESIDENT          1
        20                    5
        20 CLERK              2
        20 ANALYST            2
        20 MANAGER            1
        30                    6
        30 CLERK              1
        30 MANAGER            1

    DEPTNO JOB         COUNT(*)
---------- --------- ----------
        30 SALESMAN           4
```

ANALYSIS

Cube displays the out with all the permutation and combination of all the columns given a CUBE function.

ROLLUP FUNCTION

It is similar to that of CUBE function

INPUT
SQL> SELECT DEPTNO,JOB,COUNT(*) FROM EMP GROUP BY ROLLUP(DEPTNO,JOB)

OUTPUT

```
    DEPTNO JOB         COUNT(*)
---------- --------- ----------
        10 CLERK              1
        10 MANAGER            1
        10 PRESIDENT          1
        10                    3
        20 CLERK              2
        20 ANALYST            2
        20 MANAGER            1
        20                    5
        30 CLERK              1
        30 MANAGER            1
        30 SALESMAN           4
        30                    6
                             14
```

HAVING CLAUSE

Whenever we are using a group function in the condition, we have to use having clause. Having clause is used along with group by clause.

For example, to display Designation wise total salaries

INPUT
SQL> SELECT JOB,SUM(SAL) FROM EMP GROUP BY JOB;
OUTPUT

```
SQL> SELECT JOB,SUM(SAL) FROM EMP GROUP BY JOB;

JOB         SUM(SAL)
--------- ----------
ANALYST         6000
CLERK           4150
MANAGER         8275
PRESIDENT       5000
SALESMAN        5600
```

To Display only those designations, whose total salary is more than 5000

INPUT

SQL> SELECT JOB,SUM(SAL) FROM EMP WHERE SUM(SAL) > 5000 GROUP BY JOB;

OUTPUT

```
SELECT JOB,SUM(SAL) FROM EMP WHERE SUM(SAL) > 5000 GROUP BY JOB
                                   *
ERROR at line 1:
ORA-00934: group function is not allowed here
```

ANALYSIS

Where clause doesn't allow using group function in the condition.
When we are using group function in the condition, we have to use having clause.

INPUT

SQL> SELECT JOB,SUM(SAL) FROM EMP GROUP BY JOB HAVING SUM(SAL) > 5000;

OUTPUT

```
JOB         SUM(SAL)
--------- ----------
ANALYST         6000
MANAGER         8275
SALESMAN        5600
```

ANALYSIS

Displays all the designations whose total salary is more than 5000.

INPUT

```
SQL> SELECT JOB,COUNT(*) FROM EMP GROUP BY JOB HAVING
COUNT(*) BETWEEN 3 AND 5;
```

OUTPUT

```
JOB         COUNT(*)
--------- ----------
CLERK              4
MANAGER            3
SALESMAN           4
```

ANALYSIS

Displays all the designations whose number where employees between 3 and 5

INPUT

```
SQL> SELECT SAL FROM EMP GROUP BY SAL HAVING COUNT(SAL) > 1;
```

OUTPUT

```
       SAL
----------
      1250
      3000
```

ANALYSIS

Displays all the salaries, which are appearing more than one time in the table.

POINTS TO REMEMBER

- **WHERE clause can be used to check for conditions based on values of columns and expressions but not the result of GROUP functions.**
- **HAVING clause is specially designed to evaluate the conditions that are based on group functions such as SUM , COUNT etc.**
- **HAVING clause can be used only when GROUP BY clause is used.**

ORDER OF EXECUTION

Here are the rules ORCALE uses to execute different clauses given in SELECT command

- **Selects rows based on Where clause**
- **Groups rows based on GROUP BY clause**
- **Calculates results for each group**
- **Eliminate groups based on HAVING clause**
- **Then ORDER BY is used to order the results**

Example

```
INPUT
SQL> SELECT JOB,SUM (SAL) FROM EMP WHERE JOB != ‘CLERK’
        GROUP BY JOB HAVING SUM(SAL) > 5000  ORDER BY JOB DESC;
```

EXERCISE

ANNEXURE – A   QUERY 2

Nested Sub queries

Nesting is the act of embedding a sub query within another sub query.
SYNTAX

```
Select * FROM SOMETHING WHERE (SUBQUERY (SUBQUERY
(SUBQUERY)));
```

Whenever particular information is not accessible through a single query, then we have to write different queries one included in another.

Sub queries can be nested as deeply as your implementation of SQL allows.

We can write different types sub queries

- Single row sub queries
- Multi row sub queries
- Multi column sub queries
- Correlated sub queries.

Single row sub query
A Sub query which returns only one value.

For example,

To get the employee, who is drawing maximum salary?

INPUT

```
SQL> SELECT ENAME,SAL FROM EMP WHERE SAL = ( SELECT
        MAX(SAL) FROM EMP);
```

OUTPUT

```
ENAME          SAL
------------       ----------
KING             5000
```

ANALYSIS

Right side query is called as child query and left side query is called parent query. In nested queries, child query executes first before executing parent query.

INPUT

```
SQL> SELECT ENAME, HIREDATE FROM EMP WHERE HIREDATE = (
SELECT   MAX(HIREDATE) FROM EMP);
```

OUTPUT

```
ENAME     HIREDATE
---------- ---------
ADAMS     12-JAN-83
```

ANALYSIS

Display the least experienced employee

INPUT

```
SQL> SELECT ENAME,SAL FROM EMP WHERE SAL < (SELECT
MAX(SAL) FROM EMP);
```

OUTPUT

```
ENAME                 SAL
---------- ----------
SMITH                 800
ALLEN                1600
WARD                 1250
JONES                2975
MARTIN               1250
BLAKE                2850
CLARK                2450
SCOTT                3000
TURNER               1500
ADAMS                1100
JAMES                 950
FORD                 3000
MILLER               1300
```

ANALYSIS

Display all the employees whose salary is less than the maximum salary of all the employees.

Query

To display all the employees whose salary lines between minimum and maximum salaries

INPUT

```
SQL> SELECT * FROM EMP WHERE SAL BETWEEN (SELECT MIN(SAL)
FROM EMP) AND (SELECT MAX(SAL) FROM EMP);
```

Display all the employees who are getting maximum commission in the organization

```
SQL>  SELECT * FROM EMP WHERE COMM =
(SELECT MAX(COMM) FROM EMP);
```

Query

Display all the employees from department 30 whose salary is less than maximum salary of department 20.

```
SQL> SELECT EMPNO,ENAME,SAL FROM EMP WHERE DEPTNO=30
       AND SAL < (SELECT MAX (SAL) FROM EMP WHERE DEPTNO = 20);
```

Multi row Sub queries

A sub query, which returns more than one value.

```
INPUT
SQL>SELECT ENAME,SAL FROM EMP WHERE SAL IN(SELECT
   SAL FROM EMP GROUP BY SAL HAVING COUNT(*)> 1);

OUTPUT
ENAME          SAL
---------- -      ---------
WARD          1250
MARTIN        1250
SCOTT          3000
FORD           3000
```

ANALYSIS
Displays all the employees who are drawing similar salaries

When child query returns more than one value, we have to use IN operator for comparison.

Multi Column Sub Queries

When sub queries returns values from different columns.

```
SQL> SELECT EMPNO,ENAME,DEPTNO,SAL FROM EMP WHERE (DEPTNO,SAL)
     IN (SELECT DEPTNO,MAX(SAL) FROM EMP GROUP BY DEPTNO);
```

OUTPUT

```
     EMPNO ENAME          DEPTNO        SAL
---------- ---------- ---------- ----------
      7839 KING               10       5000
      7788 SCOTT              20       3000
      7902 FORD               20       3000
      7698 BLAKE              30       2850
```

ANALYSIS
Display all the employees who are drawing maximum salaries in each department

DML STATEMENTS IN SUB QUERIES

To modify the salary of an employee who is drawing minimum salary with the salary of the employee who is drawing maximum salary

INPUT

```
SQL> UPDATE EMP SET SAL = (SELECT MAX(SAL) FROM EMP) WHERE
     EMPNO = (SELECT EMPNO FROM EMP WHERE SAL =
     (SELECT MIN (SAL) FROM EMP));
```

ANALYSIS

Identify the employee who is drawing minimum salary and update with the maximum salary of all the employees.

To insert selected rows from emp table to emp1 table

INPUT

```
SQL> INSERT INTO EMP1
         SELECT * FROM EMP ;
```

ANALYSIS
EMP1 is an existing table. Inserts all the selected rows into EMP1 table.

CORRELATED SUB QUERIES

A correlated sub query is a sub query that receives a value from the main query and then sends a value back to main query.

For example
Display all the employees whose salary is less than maximum salary of each department

```
SQL> SELECT EMPNO,ENAME,DEPTNO,SAL FROM EMP X WHERE SAL < (SELECT MAX(SAL)
     FROM EMP WHERE DEPTNO = X.DEPTNO);

     EMPNO ENAME          DEPTNO        SAL
---------- ---------- ---------- ----------
      7369 SMITH              20        800
      7499 ALLEN              30       1600
      7521 WARD               30       1250
      7566 JONES              20       2975
      7654 MARTIN             30       1250
      7782 CLARK              10       2450
      7844 TURNER             30       1500
      7876 ADAMS              20       1100
      7900 JAMES              30        950
      7934 MILLER             10       1300
```

ANALYSIS
Find department wise maximum salaries and display the employees whose salary is less than that value for each department

Execution Sequence of steps in Correlated sub queries

- A row from main query is retrieved
- Executes sub query with the value retrieved from main query
- Sub query returns a value to main query
- Main query's current row is either selected or not, depending upon the value passed by sub query.
- This continues until all rows of main query are retrieved

To display the nth highest paid employee

```
INPUT

SQL> SELECT EMPNO,ENAME,SAL FROM EMP X WHERE &N =
     (SELECT COUNT(DISTINCT SAL) FROM EMP WHERE SAL >=X.SAL);

OUTPUT

     EMPNO ENAME             SAL
---------- ---------- ----------
      7788 SCOTT            3000
      7902 FORD             3000
```

ANALYSIS
It selects each row from emp table from parent query and finds the distinct count for each salary whose salary >= the salary returned by main query.

Least Efficient (156 Mill Sec)

```
SQL>SELECT SYSTIMESTAMP FROM DUAL;
SQL>SELECT * FROM EMP E WHERE SAL >=5000 AND JOB = 'MANAGER' AND
        25 < (SELECT COUNT(*) FROM EMP WHERE MGR = E.EMPNO) ;
SQL> SELECT SYSTIMESTAMP FROM DUAL;
```

Most Efficient (110 Mill Sec)

```
SQL>SELECT SYSTIMESTAMP FROM DUAL;

SQL>SELECT * FROM EMP E WHERE
        25 < (SELECT COUNT(*) FROM EMP WHERE MGR = E.EMPNO)  and SAL
              >=5000 AND JOB = 'MANAGER'

SQL> SELECT SYSTIMESTAMP FROM DUAL;
```

Table joins should be written first before any condition of WHERE clause. And the conditions which filter out the maximum records should be placed at the end after the joins as the parsing is done from **BOTTOM** to **TOP.**

ANY And ALL Operators

Both are used for comparing one value against a set of values.

The operator can be any one of the standard relational operators
(=, >=, >, <. <= , !=) and list is a series of values.

SYNTAX

Operator ANY list
Operator ALL list

INPUT

```
SQL> SELECT EMPNO,ENAME,SAL FROM EMP WHERE SAL > ANY(SELECT SAL FROM EMP);
```

OUTPUT

```
     EMPNO ENAME             SAL
---------- ---------- ----------
      7499 ALLEN            1600
      7521 WARD             1250
      7566 JONES            2975
      7654 MARTIN           1250
      7698 BLAKE            2850
      7782 CLARK            2450
      7788 SCOTT            3000
      7839 KING             5000
      7844 TURNER           1500
      7876 ADAMS            1100
      7900 JAMES             950
      7902 FORD             3000
      7934 MILLER           1300
```

ANALYSIS

>ANY displays greater than any values in the list.

INPUT

```
SQL> SELECT EMPNO,ENAME,SAL FROM EMP WHERE SAL < ANY (SELECT SAL FROM
EMP);
```

OUTPUT

```
     EMPNO ENAME             SAL
---------- ---------- ----------
      7369 SMITH             800
      7499 ALLEN            1600
      7521 WARD             1250
      7566 JONES            2975
      7654 MARTIN           1250
      7698 BLAKE            2850
      7782 CLARK            2450
      7788 SCOTT            3000
      7844 TURNER           1500
      7876 ADAMS            1100
      7900 JAMES             950
      7902 FORD             3000
      7934 MILLER           1300
```

ANALYSIS

Less than ANY of the list of values

INPUT

```
SQL> SELECT EMPNO,ENAME,SAL FROM EMP WHERE SAL >ALL(SELECT SAL FROM
EMP);
```

OUTPUT

```
no rows selected
```

ANALYSIS

Greater than Maximum of list

INPUT

```
SQL> SELECT EMPNO,ENAME,SAL FROM EMP WHERE SAL >ALL(3000,2000,4000);
```

OUTPUT

```
     EMPNO ENAME             SAL
---------- ---------- ----------
      7839 KING             5000
```

ANALYSIS

Greater than maximum of List

INPUT

```
SQL> SELECT EMPNO,ENAME,SAL FROM EMP WHERE SAL <ALL(3000,2000,4000);
```

OUTPUT

```
     EMPNO ENAME             SAL
---------- ---------- ----------
      7369 SMITH             800
      7499 ALLEN            1600
      7521 WARD             1250
      7654 MARTIN           1250
      7844 TURNER           1500
      7876 ADAMS            1100
      7900 JAMES             950
      7934 MILLER           1300
```

ANALYSIS
Less than minimum of List

EXISTS And NOT EXISTS Operators

These two operators are exclusively used in correlated sub query. EXISTS checks whether any row is existing in the sub query, and NOT EXISTS does the opposite.

EXISTS is different from other operators like IN , ANY etc, because it doesn't compare values of columns, instead. It checks any row is retrieved from sub query or not. If any row is retrieved from sub query the EXISTS returns true otherwise it returns False.

INPUT

```
SQL> SELECT EMPNO,ENAME,SAL,MGR FROM EMP X WHERE EXISTS(SELECT MGR FROM EMP
     WHERE X.MGR = EMPNO);
```

OUTPUT

```
     EMPNO ENAME             SAL        MGR
---------- ---------- ---------- ----------
      7369 SMITH             800       7902
      7499 ALLEN            1600       7698
      7521 WARD             1250       7698
      7566 JONES            2975       7839
      7654 MARTIN           1250       7698
      7698 BLAKE            2850       7839
      7782 CLARK            2450       7839
      7788 SCOTT            3000       7566
      7844 TURNER           1500       7698
      7876 ADAMS            1100       7788
      7900 JAMES             950       7698
      7902 FORD             3000       7566
      7934 MILLER           1300       7782
```

Note :Use **EXISTS** in place of **IN** for Base Tables to improve the performance.

Remember

The following important points to be remembered while dealing with sub queries

- **Sub query can not use ORDER BY clause. Because ORDER BY clause must be the last clause of SELECT**
- **BETWEEN … AND operator can not be used with Sub queries**

Exercise

- In department 20, one employee is drawing minimum salary and is having some designation. Display the employees from other departments whose designation is matching with the designation of the above employee.
- Display all the employees whose salary is within ±1000 from the average salary of all the employees.
- Display the employees who reported to KING
- Display all the employees whose salary is less than the minimum salary of MANAGERS.
- Display the details of students who have paid the highest amount so far in their course.
- Display the details of subjects that have been taken by more than two students

INTEGRITY CONSTRAINTS

Constraints are used to implement standard rules such as uniqueness in the key filed and business rule such as AGE column should contain a value between 15 and 60 etc.

Oracle server makes sure that the constraints are not violated whenever a row is inserted, deleted or updated. If constraint is not satisfied the operation will fail.

Constraints are normally defined at the time of creating table. But it is also possible to define constraints after the table is created.

Constraint Guidelines

- **Name a constraint or the Oracle server generates a name by using the SYS_Cn format**
- **Create a constraint either:**
  - **At the same time as the table is created, or**
  - **After the table has been created.**

- **Define a constraint at the column or table level.**
- **View a constraint in the Data**

TYPES OF CONSTRAINTS

Constraints are classified into two types

- Table Constraints
- Column Constraints

Table Constraint A constraint given at the table level is called as Table Constraint. It may refer to more than one column of the table.

A typical example is PRIMARY KEY constraint that is used to define composite primary key.

Column Constraint A constraint given at the column level is called as Column constraint. It defines a rule for a single column. It cannot refer to column other than the column, at which it is defined,
A typical example is PRIMARY KEY constraint when a single column is the primary key of the table.

Various types of Integrity constraints

- PRIMARY KEY
- UNIQUE
- NOT NULL
- CHECK

PRIMARY KEY It is used to uniquely identify rows in a table. There can be only one primary key in a table. It may consist of more than one column, if so, it is called as composite primary key. (It maintains uniqueness in the data and null values are not acceptable).

i.e. UNIQUE + NOT NULL = PRIMARY KEY

- Automatically creates unique index to enforce uniqueness.

UNIQUE Maintains unique and NULL values are acceptable.

- Oracle automatically creates a unique index for the column.

Example : EmailID

A UNIQUE key integrity constraint requires that every value in a column or set of columns (key) be unique- that is, no two rows of table can have duplicate values in a specified column or set of columns. The column (or set of columns) included in the definition of the UNIQUE key constraint is called the unique key. If the UNIQUE constraint comprises more than one column, the group of columns is called a composite unique key.

NOT NULL Uniqueness not maintained and null values are not acceptable.

Note: The NOT NULL constraint can be specified only at the column level, not at the table level.

CHECK Defines the condition that should be satisfied before insertion and updating is done.

- Defines a condition that each row must satisfy
- The following expressions are not allowed
  - References to CURRVAL, NEXTVAL and ROWNUM pseudocolumns
  - Calls to SYSDATE,UID,USER functions
  - Queries that refer to other values in other rows

Note: - Pseudocolumns are not actual columns in a table but they behave like columns. For example, you can select values from pseudocolumns. However, you cannot insert into, update, or delete from a pseudocolumn.

Guidelines for Primary Keys and Foreign Keys

• You cannot use duplicate values in a primary key.
• Primary keys generally cannot be changed.
• Foreign keys are based on data values and are purely logical, not physical, pointers.
• A foreign key value must match an existing primary key value or unique key value, or else be null.
• A foreign key must reference either a primary key or unique key column.

DDL (Data Definition language)
Create, Alter, Drop

DDL STATEMENTS COMMITS AUTOMATICALLY. There is no need to save explicitly.

Create Table

```
CREATE TABLE <TABLE-NAME> (COLUMN
DEFINITION1, COLUMN DEFINITION2);
```

Syntax :-

Column Def :
 <Name> Data type [Default Value] [constraint <name> constraint type]

Note: = Min. Column in a table = 1
 Max. Columns in a table = 1000

Rules: -

1. A table or a column name must never start a number but they can contain numbers in them
2. They can't consist of any special characters other than "$", "#", "-"
 i.e. $,# are used mainly for system tables.

Example :

```
SQL>CREATE TABLE EMPL47473 (EMPNO NUMBER (3) CONSTRAINT
     PK_EMPL47473_EMPNO PRIMARY KEY, ENAME VARCHAR2 (10)
    NOT NULL, GENDER  CHAR(1) CONSTRAINT
    CHK_EMPL47473_GENDER  CHECK(UPPER (GENDER) IN
    ('M','F')),  EMAIL_ID VARCHAR2 (30) UNIQUE, DESIGNATION
    VARCHAR2 (15), SALARY NUMBER (7,2) CHECK (SALARY
    BETWEEN 10000 AND 70000));
```

Note :

- Constraint name is useful for manipulating the given constraint
- When the constraint name is not given at the time of defining constraints, system creates a constraint with the name SYS_Cn.
- Constraints defined on a particular table are store in a data dictionary table USER_CONSTRAINTS, USER_CONS_COLUMNS.
- Tables defined by a user are stored in a data dictionary table USER_TABLES

```
SQL> DESCRIBE  USER_CONSTRAINTS
SQL> SELECT CONSTRAINT_NAME,   CONSTRAINT_TYPE,
        SEARCH_CONDITION FROM USER_CONSTRAINTS
        WHERE TABLE_NAME = 'EMPL47473';
```

OUTPUT

```
CONSTRAINT_NAME           CONSTRAINTTYPE           SEARCH_CONDITION
------------------------------ -           ----------------------------------------------------------------------------
SYS_C003018                            C                            "ENAME" IS NOT NULL

CHK_EMPL47473_GENDER                   C                     UPPER (GENDER) IN ('M','F')

SYS_C003020                            C          SALARY    BETWEEN 10000 AND 70000

PK_EMPL47473_EMPNO                     P


SYS_C003022                            U
```

ANALYSIS

Describe displays structure of the data dictionary table.

```
SQL> DESCRIBE  USER_CONS_COLUMNS
SQL> SELECT CONSTRAINT_NAME,COLUMN_NAME FROM
         USER_CONS_COLUMNS WHERE TABLE_NAME =
         'EMPL47473';
```

OUTPUT

```
CONSTRAINT_NAME                              COLUMN_NAME
------------------------------              - --------------------------------
CHK_EMPL47473_GENDER                              GENDER

PK_EMPL47473_EMPNO                                EMPNO

SYS_C003018                                       ENAME

SYS_C003020                                       SALARY

SYS_C003022                                       EMAIL_ID
```

ANALYSIS

ALTER TABLE

Used to modify the structure of a table

SYNTAX

```
ALTER TABLE <TABLENAME> [ ADD | MODIFY | DROP |
RENAME]    ( COLUMN(S));
```

ADD - for adding new columns into the table
MODIFY - for modifying the structure of columns
DROP - for removing a column in the table (8i)
RENAME - for renaming the column name (Only from 9i)

```
SQL> ALTER TABLE EMPL47473 ADD (ADDRESS
        VARCHAR2 (30), DOJ DATE,PINCODE VARCHAR2(7));
SQL> ALTER TABLE EMPL47473 MODIFY (ENAME
        CHAR (15), SALARY NUMBER (8,2));
SQL>  ALTER TABLE EMPL47473 DROP COLUMN
         PINCODE;
SQL> ALTER TABLE EMPL47473 DROP
        (DESIGNATION,ADDRESS);

  SQL> ALTER TABLE EMPL47473 RENAME COLUMN
         ENAME TO EMPNAME
```

Note: This command is also useful for manipulating constraints

INPUT

```
SQL> ALTER TABLE EMPL47473 DROP PRIMARY KEY;
```

ANALYSIS

To remove the primary key from table. Other constraints are removed only by referring constraint name.

INPUT

```
SQL>ALTER TABLE EMPL47473 ADD PRIMARY KEY(EMPNO);
```

ANALYSIS

To add primary key in the table with out constraint name. It creates constraint name with SYS_Cn.

INPUT
SQL>ALTER TABLE EMPL47473 ADD CONSTRAINT PK_EMPL47473_EMPNO PRIMARY KEY(EMPNO);

ANALYSIS
To add primary key in the table with constraint name

DATA MANIPULATION

INSERTING ROWS

SYNTAX
INSERT INTO TABLENAME [COLUMNNAME,COLUMNNAME,….]

```
SQL> INSERT INTO EMPL47473 VALUES(101,'RAVI','M',
         'RAMESH_B@YAHOO.COM',5000,'10-JAN-2001');
                              OR
SQL> INSERT INTO EMPL47473 VALUES(&EMPNO ,
'&EMPNAME','&GENDER','&EMAIL_ID',&SALARY,'&DOJ');
```

TO INSERT SPECIFIED COLUMNS IN THE TABLE

```
SQL> INSERT INTO EMPL47473(EMPNO,EMPNAME,SALARY)
          VALUES(101,'RAVI', 5000);
                              OR
SQL> INSERT INTO EMPL47473(EMPNO,EMPNAME,SALARY)
          VALUES(&EMPNO,'7EMPNAME',&SALARY);
```

ANALYSIS
We can't skip primary key and NOT NULL columns

Note :- Changes made on the database are recorded only in the shadow page. For saving the information we have to use a command COMMIT, ROLLBACK.SAVEPOINT (Called as Transactional processing statements)

SQL>**COMMIT;**

ANALYSIS

Information from shadow page flushed back to the table and shadow page gets destroyed automatically.

SQL>**ROLLBACK;**

ANALYSIS

Shadow page destroys automatically without transferring the information back to the table.

SAVEPOINT

We can use save points to roll back portions of your current set of transactions

For example

```
SQL> INSERT INTO EMPL47473 VALUES(105,’KIRAN’,’M’,
        ‘KIRAN_B@YAHOO.COM’,5000,’10-JAN-2001’);

SQL> SAVEPOINT A
```

```
SQL> INSERT INTO EMPL47473 VALUES(106,’LATHA’,’F’,
        ‘LATHA_D@YAHOO.COM’,5000,’15-JAN-2002’);

SQL> SAVEPOINT B
SQL> INSERT INTO EMPL47473 VALUES(107,'RADHA','F',
     'RADHA_V@GMAIL.COM',15000,'15-JAN-2002');
```

When we SELECT data from the table

```
SQL> SELECT * FROM EMPL47473;

     EMPNO EMPNAME    G EMAIL_ID               SALARY DOJ
---------- ---------- - ------------------ ---------- ---------
       105 KIRAN      M KIRAN_B@YAHOO.COM        5000 10-JAN-01
       106 LATHA      F LATHA_D@YAHOO.COM        5000 15-JAN-02
       107 RADHA      F RADHA_V@GMAIL.COM       15000 15-JAN-02
```

The output shows the three new records we've added . Now roll back just the last insert:

```
SQL> ROLLBACK TO B;
```

The actions that will force a commit to occur, even without your instructing it to, or quit, exit (the equivalent to exit), any DDL command forces a commit.

AUTO ROLLBACK

If you've completed a series of inserts, updates or deletes, but not yet explicitly or implicitly committed them, and you experience serious difficulties, such as a computer failure, Oracle automatically roll back any uncommitted work. If the machine or database goes down, it does this as cleanup work the next time the database is brought back up.

Note :

- Rollback works only on uncommitted data
- A DDL transaction after a DML transaction, automatically commits.
- We can use an Environment command SET VERIFY OFF to remove the old and new messages while inserting data.

ISSUE Frequent commit statements

Whenever possible, issue frequent COMMIT statements in all your programs. By issuing frequent COMMIT statements, the **performance** of the program is **enhanced** & its resource requirements are minimized as **COMMIT frees** up the following **resources:**

- Information held in the rollback segments to undo the transaction if necessary.
- All locks acquired during statement processing
- Space in the redo log buffer cache
- Overhead associated with any internal Oracle mechanisms to manage the resources in the previous three items.

CREATING A TABLE FROM ANOTHER TABLE

```
SYNTAX
CREATE TABLE <TABLENAME> AS SELECT <COLUMNS> FROM
 <EXISTING TABLE>  [WHERE <CONDITION>];
```

Example

```
SQL> CREATE TABLE EMP47473 AS SELECT EMPNO,ENAME,SAL,JOB
     FROM  EMP;
```

To add a new column in the table

```
SQL> ALTER TABLE EMP47473 ADD(SEX CHAR(1));
SQL> SELECT * FROM EMP47473;
```

UPDATING ROWS

This command is used to change the data of the table

```
SYNTAX

UPDATE <TABLENAME> SET column1 = expression, column2 =
expression WHERE <condition>;
```

```
SQL> UPDATE EMP47473 SET SAL = SAL*1.1;
SQL>  COMMIT / ROLLBACK;
```

ANALYSIS

To give uniform increments to all the employees

```
SQL> UPDATE EMP47473 SET SAL = DECODE (JOB,’CLERK’,SAL*1.1,
     ‘SALESMAN’,SAL*1.2,SAL*1.15);
SQL> COMMIT / ROLLBACK;
SQL> UPDATE EMP47473 SET SEX = ‘M’ WHERE ENAME IN
      (‘KING’,’MILLER’,’BLAKE’);
SQL> COMMIT / ROLLBACK;
SQL> SELECT * FROM EMP47473;
```

```
SQL> UPDATE EMP47473 SET SEX = ‘F’ WHERE SEX IS NULL;
SQL> COMMIT / ROLLBACK;
SQL> SELECT * FROM EMP47473 ;
```

```
SQL> UPDATE EMP47473 SET ENAME =
        DECODE(SEX,’M’,’Mr.’||ENAME,’Ms.’||ENAME);
SQL> COMMIT / ROLLBACK;
ANALYSIS
ADD Mr. or Ms. Before the existing name as per the SEX value
```

DELETING ROWS

-

SYNTAX

```
DELETE FROM <TABLENAME>  WHERE <CONDITION>;
```

```
SQL> DELETE FROM EMP47473 WHERE SEX = ‘M’;
SQL> COMMIT | ROLLBACK;
```

TRUNCATING A TABLE

SYNTAX

```
TRUNCATE TABLE <TABLENAME>
```

Note : Removes all the rows from table. Deleting specified rows is Not possible. Once the table is truncated, it automatically commits. It is a DDL statement

Droping a table

SYNTAX

```
DROP TABLE <TABLENAME>
```

Note : Table is dropped permanently. It is a DDL statement.
It removes the data along with table definitions and the table.

Adding comments to a table

You can add comments up to 2000 bytes about a column, table, view by using the **COMMENT** statement. The comment is stored in the data dictionary and can be viewed in one of the following data dictionary views in the COMMENTS column:

- ALL_COL_COMMENTS
- USER_COL_COMMENTS
- ALL_TAB_COMMENTS
- USER_TAB_COMMENTS

Syntax

COMMENT ON TABLE table | COLUMN table.column IS ‘text’ ;

REFERENTIAL INTEGRITY CONSTRAINTS

This constraint is useful for maintaining relation with other table.

Various referential integrity constraints we can use in Oracle are

- Foreign Key
- References
- On delete cascade
- On Delete Set NULL

Foreign Key: Defines the column in the child table at the table constraint level

References Identifies the table and column in the parent table

Reference key accepts NULL and duplicate values.

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.

Example

Department47473 (Deptno , dname)

Employee47473 (Empno, ename, salary, dno)

Deptno of Department47473 is a primary key
Empno of Employee47473 is a primary key
Dno of Employee47473 is a reference key

Solution

```
SQL>Create table department47473 (deptno number(3) primary key, dname
varchar2(20) Not null);

SQL>Create table employee47473(empno number(3) primary key, ename
varchar2(10)
Not null, salary number(7,2) check(salary > 0), dno number(3) references
department47473(deptno) on delete cascade);
```

Assume the case where supermarket selling various items and customers order the items.

```
SQL>Create table itemmaster (itemno number (3) primary key, itemname varchar2 (10), stock number (3) check (stock > 0));

SQL>Create table itemtran (trnno number (3), itemno number (3) references itemmaster (itemno), trndate date, trntype char (1) check (upper (trntype) in ('R','I')), quantity number (3) check (stock > 0), primary key (trnno, itemno));
```

Itemmaster

Itemno itemname stock

Itemtran

Trnno itemno trndate trntype quantity

Assume the case where with each transaction

```
ALTER TABLE <TABLENAME> DISABLE PRIMARY KEY

ALTER TABLE <tablename> DISABLE PRIMARY KEY CASCADE;
```

Note : It is not possible to enable using cascade

```
DROP TABLE <TABLENAME> CASCADE CONSTRAINTS
```

ANALYSIS

Dropping the table along with constraints

```
ALTER TABLE <tablename> DROP PRIMARY KEY CASCADE;
```

ANALYSIS

Removing the primary key along with Reference key

Exercise

- Consider a training institute conducting different courses, into which the students are joining for various courses (Also, assume the case where same student can join in more than one course)
- The students may pay the fee in installments

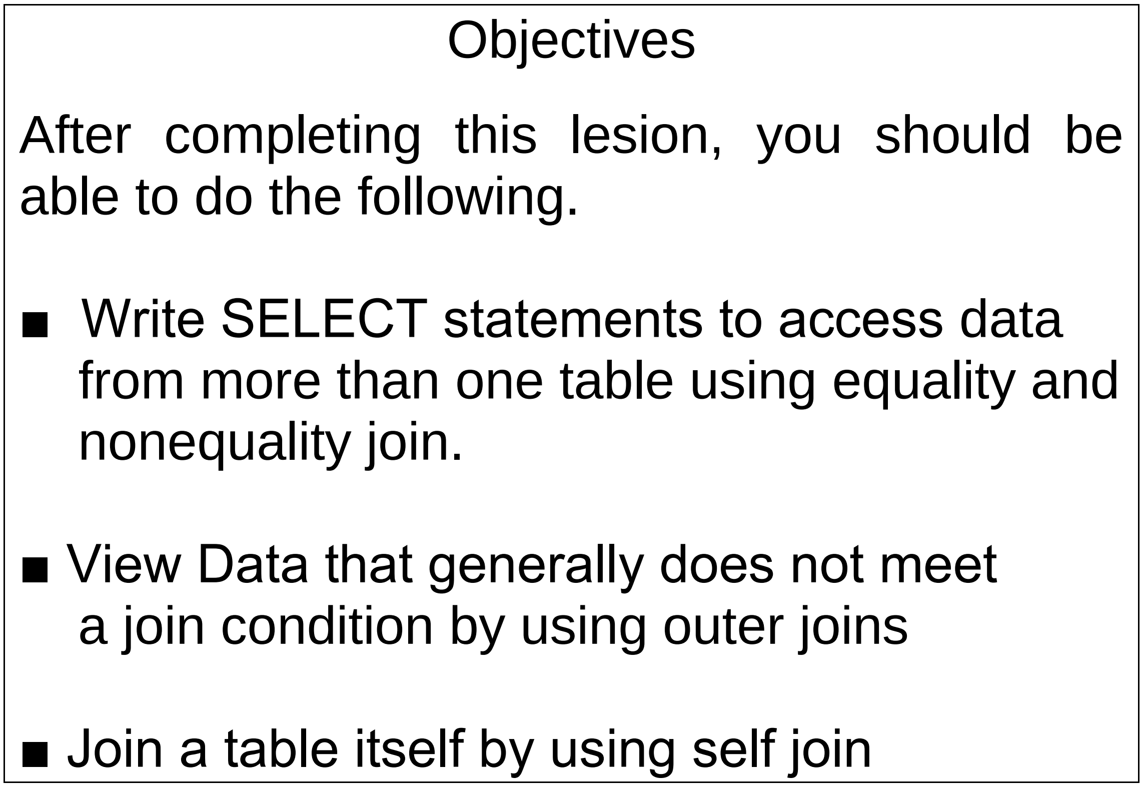
Identify the tables, attributes and define them with relations

JOINS

Objectives

Join will enable you to gather and manipulate data across several tables. By

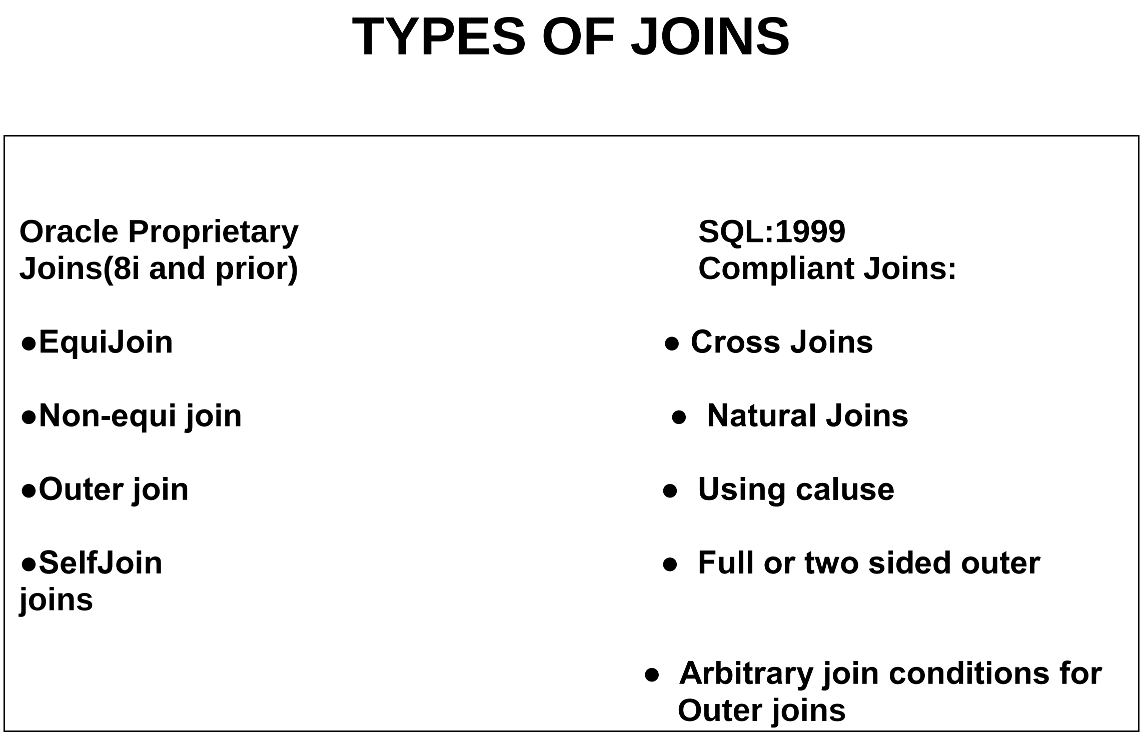
One of the most powerful features of SQL is its capability to gather and manipulate data from across several tables. Without this feature you would have to store all the data elements necessary for each application in one table. Without common tables you would need to store the same data in several tables.



Objectives

After completing this lesion, you should be able to do the following.

- Write SELECT statements to access data from more than one table using equality and nonequality join.
- View Data that generally does not meet a join condition by using outer joins
- Join a table itself by using self join



TYPES OF JOINS

Oracle Proprietary Joins(8i and prior)	SQL:1999 Compliant Joins:
●EquiJoin	● Cross Joins
●Non-equi join	● Natural Joins
●Outer join	● Using caluse
●SelfJoin joins	● Full or two sided outer
	● Arbitrary join conditions for Outer joins

Types of Joins

The Oracle 9i database offers join syntax that is SQL: 1999 compliant. Prior to 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.

Joining tables using Oracle Syntax

Use a join to query data from more than one table

```
SELECT      Table1.column1, table2.column2

FROM table1,table2

WHERE table1.column1 = table2.column2;
```

- **Write the join condition in the WHERE clause.**
- **Prefix the column name with table name when the same column name appears in more than one table.**

Guidelines

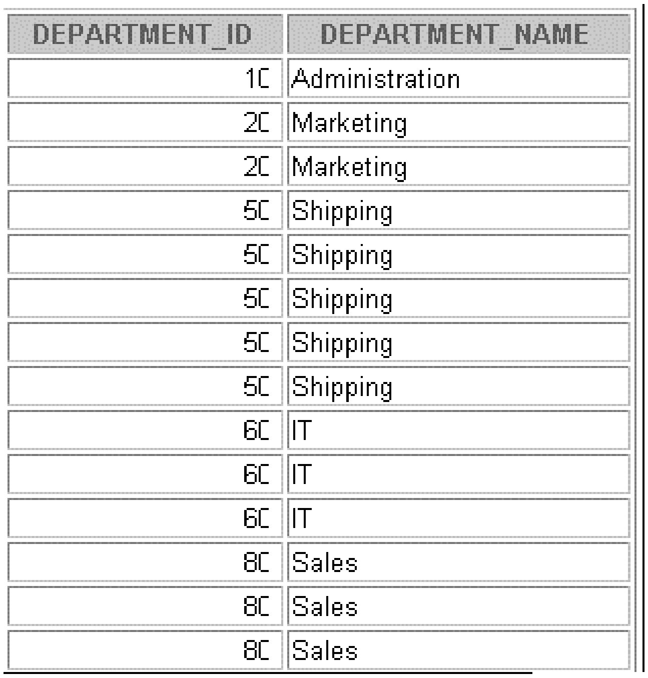
- When writing a SELECT statement that joins tables, precede the common column name with the table name for clarify and to enhance database access.

- If the same column name appears in more than one table, the column name must be prefixed with the table name.

- To join n tables together, you need a minimum of n-1 join conditions. This rule many not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row**.**

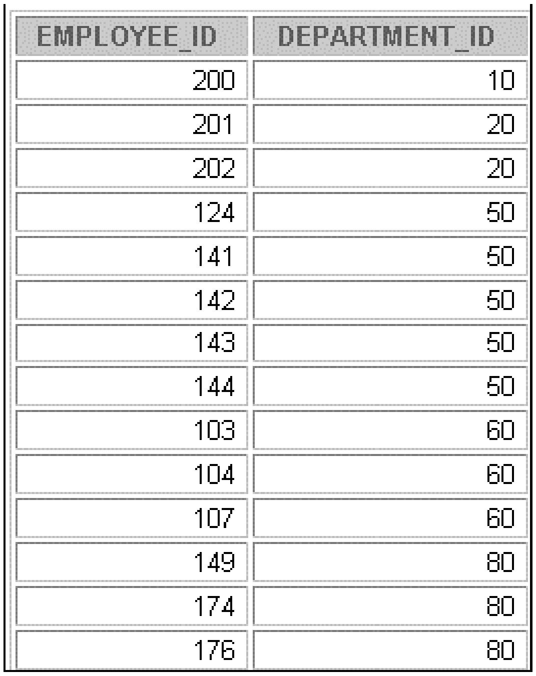
What is An Equi Join?

Departments



DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT
60	IT
80	Sales
80	Sales
80	Sales

Employees



EMPLOYEE_ID	DEPARTMENT_ID
200	10
201	20
202	20
124	50
141	50
142	50
143	50
144	50
103	60
104	60
107	60
149	80
174	80
176	80

Primary key

Foreign Key

↑

↑

Equi join

Extracting the information from more than one table by comparing (=) the common information.

Note : Equi Joins are also called as simple joins or Inner Joins

To display common column information

SQL> SELECT EMPNO,ENAME,JOB,SAL,DNAME FROM EMP,DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;

OUTPUT

```
     EMPNO ENAME      JOB              SAL DNAME
---------- ---------- --------- ---------- --------------
      7782 CLARK      MANAGER         2450 ACCOUNTING
      7839 KING       PRESIDENT       5000 ACCOUNTING
      7934 MILLER     CLERK           1300 ACCOUNTING
      7369 SMITH      CLERK            800 RESEARCH
      7876 ADAMS      CLERK           1100 RESEARCH
      7902 FORD       ANALYST         3000 RESEARCH
      7788 SCOTT      ANALYST         3000 RESEARCH
      7566 JONES      MANAGER         2975 RESEARCH
      7499 ALLEN      SALESMAN        1600 SALES
      7698 BLAKE      MANAGER         2850 SALES
      7654 MARTIN     SALESMAN        1250 SALES
      7900 JAMES      CLERK            950 SALES
      7844 TURNER     SALESMAN        1500 SALES
      7521 WARD       SALESMAN        1250 SALES
```

ANALYSIS
Efficiency is more when we compare the information from lower data table(master table) to Higher data table(child table).

When Oracle processes multiple tables, it uses an internal sort/merge procedure to join those tables. First, it scans & sorts the first table (the one specified last in FROM clause). Next, it scans the second table (the one prior to the last in the FROM clause) and merges all of the retrieved from the second table with those retrieved from the first table. It takes around 0.96 seconds

SQL> SELECT EMPNO,ENAME,JOB,SAL,DNAME FROM DEPT,EMP
WHERE EMP.DEPTNO = DEPT.DEPTNO;

ANALYSIS

Here driving table is EMP. It takes around 26.09 seconds
So, Efficiency is less.

Non-Equi joins

Getting the information from more than one table without using comparison (=) operator.

```
INPUT
SQL> select empno,ename,sal,grade,losal,hisal from salgrade g,emp e
        where e.sal between g.losal and g.hisal
/

ANALYSIS

Displays all the employees whose salary lies between any pair of low and high salary ranges.
```

```
INPUT
SQL> SELECT * FROM DEPT WHERE DEPTNO NOT IN
      (SELECT DISTINCT DEPTNO FROM EMP);
OUTPUT
   DEPTNO DNAME        LOC
    ---------- --------------     -------------
        40 OPERATIONS     BOSTON

ANALYSIS
Displays the details of the department where there are no employees
```

We can also get above output using relational algebra operators.

```
SQL> SELECT DEPTNO FROM DEPT
          MINUS
           SELECT DEOTNO FROM EMP;
```

```
SQL> SELECT DEPTNO FROM DEPT
          UNION
           SELECT DEOTNO FROM EMP;
```

```
SQL> SELECT DEPTNO FROM DEPT
         UNION ALL
          SELECT DEOTNO FROM EMP;
```

OUTER JOIN

It is a join, which forcibly joins multiple tables even without having the common information. It is represented by +.

```
SQL> SELECT EMPNO,ENAME,JOB,SAL,DNAME FROM DEPT,EMP
        WHERE DEPT.DEPTNO = EMP.DEPTNO(+);
```

OUTPUT

```
     EMPNO ENAME      JOB              SAL DNAME
---------- ---------- --------- ---------- --------------
      7782 CLARK      MANAGER         2450 ACCOUNTING
      7839 KING       PRESIDENT       5000 ACCOUNTING
      7934 MILLER     CLERK           1300 ACCOUNTING
      7369 SMITH      CLERK            800 RESEARCH
      7876 ADAMS      CLERK           1100 RESEARCH
      7902 FORD       ANALYST         3000 RESEARCH
      7788 SCOTT      ANALYST         3000 RESEARCH
      7566 JONES      MANAGER         2975 RESEARCH
      7499 ALLEN      SALESMAN        1600 SALES
      7698 BLAKE      MANAGER         2850 SALES
      7654 MARTIN     SALESMAN        1250 SALES
      7900 JAMES      CLERK            950 SALES
      7844 TURNER     SALESMAN        1500 SALES
      7521 WARD       SALESMAN        1250 SALES
                                           OPERATIONS
```

LEFT, RIGHT AND FULL OUTER JOIN

As of Oracle 9i, you can use the ANSI SQL standard syntax for outer joins. In the FROM clause, you can tell Oracle to perform a LEFT, RIGHT or FULL OUTER join.

```
SQL> SELECT EMPNO, ENAME, JOB, DEPT.DEPTNO, DNAME FROM EMP
     LEFT OUTER JOIN DEPT ON DEPT.DEPTNO = EMP.DEPTNO;

     EMPNO ENAME      JOB           DEPTNO DNAME
---------- ---------- --------- ---------- --------------
      7934 MILLER     CLERK             10 ACCOUNTING
      7839 KING       PRESIDENT         10 ACCOUNTING
      7782 CLARK      MANAGER           10 ACCOUNTING
      7902 FORD       ANALYST           20 RESEARCH
```

```
SQL> SELECT EMPNO, ENAME, JOB, DEPT.DEPTNO, DNAME FROM DEPT
     LEFT OUTER JOIN EMP ON DEPT.DEPTNO = EMP.DEPTNO;

OUTPUT
     EMPNO ENAME      JOB           DEPTNO DNAME
---------- ---------- --------- ---------- --------------
      7369 SMITH      CLERK             20 RESEARCH
      7499 ALLEN      SALESMAN          30 SALES
      7521 WARD       SALESMAN          30 SALES
      7566 JONES      MANAGER           20 RESEARCH
      7654 MARTIN     SALESMAN          30 SALES
      7698 BLAKE      MANAGER           30 SALES
      7782 CLARK      MANAGER           10 ACCOUNTING
      7788 SCOTT      ANALYST           20 RESEARCH
      7839 KING       PRESIDENT         10 ACCOUNTING
      7844 TURNER     SALESMAN          30 SALES
      7876 ADAMS      CLERK             20 RESEARCH
      7900 JAMES      CLERK             30 SALES
      7902 FORD       ANALYST           20 RESEARCH
      7934 MILLER     CLERK             10 ACCOUNTING
                                        40 OPERATIONS
```

```
SQL> SELECT EMPNO, ENAME, JOB, DEPT.DEPTNO, DNAME FROM DEPT
     RIGHT OUTER JOIN EMP ON DEPT.DEPTNO = EMP.DEPTNO;

OUTPUT
     EMPNO ENAME      JOB           DEPTNO DNAME
---------- ---------- --------- ---------- --------------
      7369 SMITH      CLERK             20 RESEARCH
      7499 ALLEN      SALESMAN          30 SALES
      7521 WARD       SALESMAN          30 SALES
      7566 JONES      MANAGER           20 RESEARCH
      7654 MARTIN     SALESMAN          30 SALES
      7698 BLAKE      MANAGER           30 SALES
      7782 CLARK      MANAGER           10 ACCOUNTING
      7788 SCOTT      ANALYST           20 RESEARCH
      7839 KING       PRESIDENT         10 ACCOUNTING
      7844 TURNER     SALESMAN          30 SALES
      7876 ADAMS      CLERK             20 RESEARCH
      7900 JAMES      CLERK             30 SALES
      7902 FORD       ANALYST           20 RESEARCH
      7934 MILLER     CLERK             10 ACCOUNTING
```

ANALYSIS

Gets the common information from both tables, and then forcibly joins from dept table to emp table.

```
SQL> SELECT EMPNO, ENAME, JOB, DEPT.DEPTNO, DNAME FROM EMP
     RIGHT OUTER JOIN DEPT ON DEPT.DEPTNO = EMP.DEPTNO;

OUTPUT
     EMPNO ENAME      JOB           DEPTNO DNAME
---------- ---------- --------- ---------- --------------
      7369 SMITH      CLERK             20 RESEARCH
      7499 ALLEN      SALESMAN          30 SALES
      7521 WARD       SALESMAN          30 SALES
      7566 JONES      MANAGER           20 RESEARCH
      7654 MARTIN     SALESMAN          30 SALES
      7698 BLAKE      MANAGER           30 SALES
      7782 CLARK      MANAGER           10 ACCOUNTING
      7788 SCOTT      ANALYST           20 RESEARCH
      7839 KING       PRESIDENT         10 ACCOUNTING
      7844 TURNER     SALESMAN          30 SALES
      7876 ADAMS      CLERK             20 RESEARCH
      7900 JAMES      CLERK             30 SALES
      7902 FORD       ANALYST           20 RESEARCH
      7934 MILLER     CLERK             10 ACCOUNTING
                                        40 OPERATIONS
```

Position of Joins in where clause

Table joins should be written first before any condition of WHERE clause. And the conditions which filter out the maximum records should be placed at the end after the joins as the parsing is done from **BOTTOM** to **TOP**

For ex;

Least Efficient (Total CPU = 153.6 Sec)

```
SELECT ENAME,JOB,MGR FROM EMP E WHERE SAL > 50000 AND
 JOB = ‘MANAGER’
AND 25 < (SELECT COUNT(*) FROM EMP WHERE MGR = E.EMPNO);
```

Most Efficient (Total CPU time = 10.6 sec)

```
SELECT ENAME,JOB,MGR FROM EMP E WHERE 25 <
  (SELECT COUNT(*) FROM EMP WHERE MGR = E.EMPNO)
AND  SAL > 5000 AND JOB = ‘MANAGER’;
```

SELF JOIN

Joining the table from itself is called as self join.

```
SQL> SELECT WORKER.ENAME || ' IS WORKING UNDER ' ||
MANAGER.ENAME FROM EMP WORKER, EMP MANAGER
WHERE WORKER.MGR = MANAGER.EMPNO;
```

OUTPUT

```
WORKER.ENAME||'ISWORKINGUNDER'||MANAGE
---------------------------------------
SCOTT IS WORKING UNDER JONES
FORD IS WORKING UNDER JONES
ALLEN IS WORKING UNDER BLAKE
WARD IS WORKING UNDER BLAKE
JAMES IS WORKING UNDER BLAKE
TURNER IS WORKING UNDER BLAKE
MARTIN IS WORKING UNDER BLAKE
MILLER IS WORKING UNDER CLARK
ADAMS IS WORKING UNDER SCOTT
JONES IS WORKING UNDER KING
CLARK IS WORKING UNDER KING
BLAKE IS WORKING UNDER KING
```

NATURAL AND INNER JOINS (Introduced in 9i)

We can use natural keyword to indicate that a join should be performed based on all columns that have the same name in the two tables being joined.

Creating Natural Joins

- The Natural join clause is based on all columns in the two tables that have the same name,
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned.

For example, To get the common information from two tables,

INPUT

```
SQL> SELECT EMPNO,ENAME,JOB,DEPTNO,DNAME FROM DEPT
  2  NATURAL JOIN EMP;
```

OUTPUT

```
     EMPNO ENAME      JOB           DEPTNO DNAME
---------- ---------- --------- ---------- --------------
      7782 CLARK      MANAGER           10 ACCOUNTING
      7839 KING       PRESIDENT         10 ACCOUNTING
      7934 MILLER     CLERK             10 ACCOUNTING
      7369 SMITH      CLERK             20 RESEARCH
      7876 ADAMS      CLERK             20 RESEARCH
      7902 FORD       ANALYST           20 RESEARCH
      7788 SCOTT      ANALYST           20 RESEARCH
      7566 JONES      MANAGER           20 RESEARCH
      7499 ALLEN      SALESMAN          30 SALES
      7698 BLAKE      MANAGER           30 SALES
      7654 MARTIN     SALESMAN          30 SALES
      7900 JAMES      CLERK             30 SALES
      7844 TURNER     SALESMAN          30 SALES
      7521 WARD       SALESMAN          30 SALES
```

Creating Joins with Using Caluse

- **If several columns have the same names but the data types do not match, the NATURAL JOIN clause can be modified with USING clause to specify the columns that should be used for an equijoin.**

- **use the USING clause to match only one column when more than one column matches.**

- **Do not use a tablename or alias in the referenced columns**

- **The NATURAL JOIN and USING clauses are mutually exclusive.**

INNER JOIN

Support for INNER join syntax was introduced in Oracle9i, inner joins are the default – they return the rows the two tables have in common, and are the alternative to outer joins. Note that they support ON clause, so that you can specify join criteria.

```
SQL>SELECT EMPNO, ENAME, JOB, DEPT.DEPTNO, DNAME FROM EMP
INNER JOIN DEPT ON EMP.DEPTNO = DEPT.DEPTNO;
```

CROSS JOIN (CARTESIAN PRODUCT)

Joining tables without giving proper join condition.

```
INPUT

SQL> SELECT * FROM DEPT,EMP;
                    OR
SQL> SELECT * FROM EMP  CROSS JOIN DEPT;

ANALYSIS

It multiplies the rows from both tables and displays the output
i.e. 14 rows (emp table) X 4 rows(dept table) = 56 rows.
```

EXERCISE

ANNEXURE   QUERY III

OTHER OBJECTS

SEQUENCE OBJECT

Used to generate sequence(Unique) Integers for use of primary keys.

SYNTAX

```
CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

`Sequence` is the name of the sequence generator

`INCREMENT BY n` specifies the interval between sequence numbers where *n* is an integer (If this clause is omitted, the sequence increments by 1.)

`START WITH n` specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)

`MAXVALUE n` specifies the maximum value the sequence can generate

`NOMAXVALUE` specifies a maximum value of 10^27 for an ascending sequence and –1 for a descending sequence (This is the default option.)

`MINVALUE n` specifies the minimum sequence value

`NOMINVALUE` specifies a minimum value of 1 for an ascending sequence and –(10^26) for a descending sequence (This is the default option.)

`CYCLE | NOCYCLE` specifies whether the sequence continues to generate values after reaching its maximum or minimum value (`NOCYCLE` is the default Option.)

`CACHE n | NOCACHE` specifies how many values the Oracle server preallocates and keep in memory (By default, the Oracle server caches 20 values.) The value set must be less than MAXVALUE minus

Example

```
CREATE SEQUENCE SQNO47473
   START WITH 1
   INCREMENT BY 1
   MAXVALUE 10;
```

```
CREATE SEQUENCE SQNO47473
   START WITH 1
   INCREMENT BY 1
   MAVALUE 10
   CACHE 3
  CYCLE;
```

Note: - These sequences are stored in a data dictionary table USER_SEQUENCES.
This sequence object provides two public member functions

NEXTVAL and CURRVAL

NEXTVAL is a function which generate next value from sequence object
CURRVAL is a function, which gives the current value of the sequence object

Assume there is a table

SAMPLE47473		
EMPNO	ENAME	SAL

To insert the values into the table

```
SQL> INSERT INTO SAMPLE47473 VALUES(SQNO47473.NEXTVAL,
        '&ENAME', &SAL);
```

TO MODIFY THE SEQUNECE OBJECT

```
SQL> ALTER SEQUENCE SQNO47473
       INCREMENT BY 2
      MAXVALUE 40;

Note : We can't change starting value
```

To remove the sequence object

```
SQL>  DROP SEQUENCE <SEQUENCE_NAME>;
```

VIEWS

- A view is an object, which is a logical representation of a table
- A view contains no data on its own
- It is derived from tables
- Changes made in tables are automatically reflected in views
- As a view does not store any data the redundancy problem does not arise.+-
- Critical data in the base table is safeguarded as access to such data can be controlled.
- It is used to reduce the complexity of the query

In Oracle Oracle we can create different types of views

SIMPLE COMPLEX INLINE

SIMPLE view is a view, which is created using only one base table.

COMPLEX view is a view, which is created using more than one table or using group functions

INLINE view is a view, which is created using sub query (it is not a schema object. • It is a named sub query in the `FROM` clause of the main query. Generally used in TOP N Analysis.

SYNTAX

```
CREATE OR REPLACE [FORCE]  VIEW <VIEWNAME> AS SELECT
<COLUMNS> FROM <TABLE > [ WITH READ ONLY];
```

The table on which a view is based is called as **base table**
FORCE option allows view to be created even if the base table doesn't exist. However, the base table should exist before the view is used.

Changing Base Table through view

A view can also be used to change the data of base table
A view can be used to delete, insert and update rows in the base table.

However for each operation certain conditions are to be satisfied.

Rules for deleting row

The following should NOT be used in the query.

- More than one table
- GROUP BY clause
- GROUP Function
- DISTINCT clause
- Pseudo column ROWNUM

Rules for updating rows

The following are the rules that are to be satisfied to update base table through view.

- All the rules of Delete are applicable
- The column being updated should not be derived from an expression.

Rules for insertion of rows

- All the rules of UPDATE
- Al NOT NULL columns should be included in the view.

Example

```
SQL> CREATE OR REPLACE VIEW TESTVIEW47473 AS SELECT
       EMPNO,ENAME,SAL FROM EMP47473;
```

Note: - These views are stored in a data dictionary table USER_VIEWS

```
SQL> CREATE OR REPLACE VIEW TESTVIEW47473 AS
SELECT
       EMPNO,ENAME,SAL FROM EMP47473 WITH READ ONLY;

ANALYSIS
```

WITH CHECK OPTION

This option is used to prevent any changes to base table through view. Insertion and updation is not allowed into base table through view.

```
SQL> CREATE OR REPLACE VIEW CHKVIEW AS SELECT * FROM EMP
        WHERE DEPTNO = 20 WITH CHECK OPTION;
```

It doesn’t allow you to update the condition column as well as it doesn’t allow you to insert the details of employees with DEPTNO other than 20.

We can also create a view using group functions. Such views are called as INLINE views. They are by default read only.

```
SQL> CREATE OR REPLACE VIEW SIMPLEVIEW47473 AS
SELECT
     SUBSTR(HIREDATE,-2) YEAR, COUNT(*) NUMB FROM EMP47473
    GROUP BY JOB;
```

To remove a view

```
SQL> DROP VIEW <VIEWNAME>;
```

INDEX

The concept indexing in Oracle is same as a book index. Just like how book index is sorted in the ascending order of topics, an index in Oracle is a list of values of a column in the ascending order. Page number in book index is similar to ROWID if Oracle index.

An oracle index is a database object. It contains the values of the indexed column(s) in the ascending order along with address of each row. The address of rows are obtained using ROWID pseudo column.

Why to Use An INDEX

INDEXES in ORACLE are used for two purposes

- To speed up data retrieval and thereby improving performance of query
- To enforce uniqueness

Note :- A UNIQUE index is automatically created when you use PRIMARY KEY and UNIQUE constraints

An index can have up to 32 columns.

```
SYNTAX
CREATE [UNIQUE] INDEX  index_name ON table(column1,column2,…);
```

Note :- Indexes are stored in the data dictionary table USER_INDEXES.

When Oracle Does Not Use Index

Oracle index is completely automatic. I.e., you never have to open or close an index. Oracle server decides whether to use an index or not.

The following are the cases in which Oracle does NOT use index.

- SELECT doesn’t contain WHERE clause
- When the data size is less
- SELECT contains WHERE clause, but WHERE clause doesn’t refer to indexed column.
- SELECT contains WHERE clause and WHERE clause uses indexed columns but indexed column is modified in the WHERE clause.

Negative Side of an Index

INDEX plays an important role in improving performance. But at the same time it may also degrade performance, if not designed carefully.

Having many indexes may have negative impact on the performance because whenever there is a change in the table, immediately index is to reflect that change. A new row's insertion will effect index and Oracle server implicitly updates index. So this will put more burden on the machine if more number of indexes are created on a table.

FUNCTIONAL INDEX

As of Oracle8i, we can create functional-based indexes. When we are storing alpha-numeric information, we may store the information in any case. When we create index on such columns, information is placed in different ranges of indexes. So, before creating index, we can convert them in to single case.

```
SQL> CREATE INDEX IDX_NAME ON EMP(UPPER(ENAME));
```

Dropping an Index

```
SYNTAX

DROP INDEX <INDEXNAME>;
```

Removing an index doesn't invalidate existing applications, because applications are not directly dependent on index, but at the same time not having an index may effects performance.

PSEUDO COLUMN

A pseudo-column is a column that yields a value when selected but which is not an actual column of the table.

Example

ROWID
ROWNUM
SYADATE
NEXTVAL
CURRVAL
NULL
LEVEL

Are called as Pseudo-columns.

SELECT ROWNUM, EMPNO, ENAME FROM EMP;

TO DISPLAY 3 HIGHEST PAID EMPLOYEES

```
SQL> SELECT ROWNUM,EMPNO,ENAME,SAL FROM (SELECT EMPNO,
     ENAME,SAL FROM EMP ORDER BY SAL DESC)
     WHERE ROWNUM <= 3;
```

Exercise

1) **Display the string SATYAM in the format**

S
A
T
Y
A
M

2) **Display only even rows from the table**
3) **Display one year calendar**
4) **Display how many a’s are there in the given string**
5) **Remove duplicate rows from the given table**

Empno	ename
1	x
2	y
3	z
1	x
3	z

6)Find out how many columns are there in a given table(Use the data dictionary table USER_TAB_COLUMNS)

ADVANCED QUERIES

ANALYTICAL QUERIES

Analytical functions are used mainly for the analysis of data as required by decision-making managers.
Oracle has embedded analytical functions in SQL statement to cater to most of the requirements of data mining.

These functions are listed below

Ranking For calculating ranks, percentiles, and n-tiles of the values in a result set.

For example, to find out top three salaried employees

```
SQL> SELECT RANK() OVER(ORDER BY SAL DESC) DEFAULT_RANK, SAL FROM EMP;

DEFAULT_RANK        SAL
------------ ----------
           1   855945.6
           1   855945.6
           1   855945.6
           4   641959.2
           5  196867.32
           6  184563.24
           7  153802.68
           7  153802.68
           9  129461.88
          10   111807.6
          11   66870.77
          12      10000
          13    6810.91
```

ANALYSIS
When different employees salary is same, they get the same rank. There is a gap between ranks. In the example, see the ranks between 1 and 4.

DENSE_RANK

The main difference between RANK () and DENSE_RANK () is that in the DENSE_RANK () there is no gap between ranks.

```
SQL> SELECT DENSE_RANK() OVER(ORDER BY SAL DESC) DEFAULT_RANK, SAL FROM EMP;

DEFAULT_RANK        SAL
------------ ----------
           1   855945.6
           1   855945.6
           1   855945.6
           2   641959.2
           3  196867.32
           4  184563.24
           5  153802.68
           5  153802.68
           6  129461.88
           7   111807.6
           8   66870.77
           9      10000
          10    6810.91
```

ROW_NUMBER

The function row number assigns unique rank to each row even if they are having the same value of order by expression. The rows will get the same rank if their order by expression has the same value.

INPUT

```
  SQL> SELECT RANK() OVER(ORDER BY SAL) DEFAULT_RANK, ROW_NUMBER() OVER(ORDER
BY SAL) RW_NUM,SAL FROM EMP
```

OUTPUT

```
DEFAULT_RANK     RW_NUM        SAL
------------ ---------- ----------
           1          1    6810.91
           2          2      10000
           3          3   66870.77
           4          4   111807.6
           5          5  129461.88
           6          6  153802.68
           6          7  153802.68
           8          8  184563.24
           9          9  196867.32
          10         10   641959.2
          11         11   855945.6
          11         12   855945.6
          11         13   855945.6
```

NULLIF FUNCTION

This function produces NULL value if the expression has a specified value. It produces NULL value if the expression has a specified value. This is like a complement of the NVL function.

INPUT

```
SQL> SELECT ENAME,JOB,COMM,NULLIF(JOB,'MANAGER') FROM EMP;
```

OUTPUT

```
ENAME           JOB             COMM NULLIF(JO
--------------- --------- ---------- ---------
ss ss           CLERK           1234 CLERK
allen           SALESMAN       746.5 SALESMAN
WARD            SALESMAN     1244.16 SALESMAN
JONES           MANAGER
MARTIN          SALESMAN     3483.65 SALESMAN
BLAKE           MANAGER
CLARK           MANAGER
SCOTT           ANALYST              ANALYST
TURNER          SALESMAN           0 SALESMAN
ADAMS           CLERK                CLERK
JAMES           CLERK                CLERK
FORD            ANALYST              ANALYST
phani           se             20000 se
```

ANALYSIS
It is just opposite to NVL(). NVL() substitute value, if its is NULL. Where as NULLIF produces NULL, if value matches.

NVL2 Function

It is an extended form of NVL.

Syntax

NVL2(expr1,expr2,expr3)

In NVL2, expr1 can never be returned; either expr2 or expr3 will be returned.

If expr1 is not NULL, NVL2 returns expr2,
If expr1 is NULL , NVL2 returns expr3.

The expr1 can have any data type. The arguments expr2 and expr3 can have any datatype except LONG.

```
INPUT

SQL> SELECT COMM,NVL2(COMM,COMM,0) FROM EMP;

OUTPUT

      COMM NVL2(COMM,COMM,0)
---------- -----------------
                           0
       300               300
       500               500
                           0
      1400              1400
                           0
                           0
                           0
                           0
         0                 0
                           0
                           0
                           0
                           0
```

Coalesce Function

This function takes n arguments and produces first argument, which is having the first NOT NULL value.

```
SQL> SELECT ENAME, COMM, SAL, DEPTNO, MGR, COALESCE (COMM, SAL, DEPTNO, MGR)
COAL FROM EMP;

ENAME                 COMM        SAL     DEPTNO        MGR       COAL
--------------- ---------- ---------- ---------- ---------- ----------
ss ss                 1234    6810.91         20       7902       1234
allen                746.5  196867.32         30       7698      746.5
WARD               1244.16  153802.68         30       7698    1244.16
JONES                        855945.6         20       7839   855945.6
MARTIN             3483.65  153802.68         30       7698    3483.65
BLAKE                        855945.6         30       7839   855945.6
CLARK                        855945.6         10       7839   855945.6
SCOTT                        641959.2         20       7566   641959.2
TURNER                   0  184563.24         30       7698          0
ADAMS                       129461.88         20       7788  129461.88
JAMES                        111807.6         30       7698   111807.6
FORD                         66870.77         20       7566   66870.77
phani                20000      10000         90       9822      20000
```

Multiple Insert and Merge statements

Oracle 9i provides enhanced facility of loading data in the table using select statement.

Using this facility you

- Load data in multiple tables in single insert statement
- Load multiple rows in the same table in single insert statement
- Load data conditionally in the table.

Multiple insert statement

There are four types of multiple inert statements

- Unconditional insert statement
- Pivoting insert statement
- Conditional insert statement
- Insert first statement

Unconditional Insert Statement

This statement enables to you insert data in multiple tables using a single insert statement.

To understand this query consider the three tables from Annexure

To insert the data into two tables

```
Example1(pname,title,salary)
Example2(pname,course,cost)

Insert all
Into example1 values(pname,title,salary)
Into example2 values(pname,course,cost)
Select p.pname,title,course,cost,salary from programmer p,software s,
studies st
Where p.pname = s.pname and s.pname = st.pname;
```

Conditional Insert statement

It inserts the data into the table only when condition is satisfied. Assume there are three tables with structure (table1,table2,table3)

Empno.ename,job,sal,deptno

We can load the data conditionally into three tables by using following statement

```
SQL> insert all
        When deptno = 10 then
        Insert into table1 values(empno,ename,job,sal,deptno);
        When deptno = 20 then
        Insert into table2 values(empno,ename,job,sal,deptno);
        When deptno = 30 then
        Insert into table3 values(empno,ename,job,sal,deptno);
        Select empno,ename,sal,deptno from emp
```

Insert First statement

```
Insert first
When deptno = 10 then
     Insert into table1 values(empno,ename,job,sal,deptno);
 When deptno = 20 then
     Insert into table2 values(empno,ename,job,sal,deptno);
  When deptno = 30 then
      Insert into table3 values(empno,ename,job,sal,deptno);
   Select empno,ename,sal,deptno from emp;
```

The difference between Insert First and Insert All is that in the formar at the most one row is inserted into the table while latter rows may be inserted into multiple tables.

LOCKING MECHANISMS

To ensure data integrity oracle uses locks. Locks are used to prevent destructive interaction between processes accessing the same resource.

Oracle uses two locking types

- DML locks - used to protect data. Can be either table or row level.
- Dictionary locks – used to protect the database structure

Locks may be explicit or implicit. A process or user instigates explicit locks. Implicit locks are undertaken by Oracle. Oracle will lock any resource when it detects a valid attempt to update the resource. Dictionary locks are always implicit. DML locks may be implicit or explicit.

There are five locking modes. They are

EXCLUSIVE(X)	LOCK ALLOWS QUERIES BUT NOTHING ELSE
SHARE(S)	Lock allows queries but not updates
ROW SHARE(RS)	Lock allows concurrent process access to table. Resource may not be locked exclusively.
ROW EXCLUSIVE (RX)	Same as row share but no share mode locking. Updates. Deletes and inserts use this lock mode.

Locks are held until either a transaction is committed / rolled back.

Locking using SELECT for UPDATE

```
SELECT * FROM EMP WHERE EMPNO = 7521 FOR UPDATE OF SAL
NOWAIT;
```

With this SELECT we lock all the rows in the result set for later update. The FOR UPDATE tells Oracle to lock each row as it processes it.

The OF keyword prefixes the column identification area which specifies which columns are going to be updated by us at a later date.

The NOWAIT keyword specifies that we don’t want the statement to wait until, and current locks on the table are removed.

Assume that there are tow users

SCOTT AND X

In the Scott user there is one table with the name EMP.
Scott has given some privileges on EMP table to X user.

If X wants to lock the EMP table, then he has to issue a command

```
LOCK TABLE SCOTT.EMP IN EXCLUSIVE MODE NOWAIT;
```

Now from SCOTT user , if he tries to the lock the table, which is already locked by X

```
LOCK TABLE EMP IN EXCLUSIVE MODE;
```

It results an error

```
LOCK TABLE EMP IN EXCLUSIVE MODE NOWAIT
      *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified
```

SECURITY

What is a privilege

A privilege is a right to access an object such as a table, view etc., or to execute a particular type of SQL command such as CREATE TABLE.

Privileges are classified into two categories depending upon what type of right they give to the user.

- Object Privileges
- System privileges

Object privilege

An object privilege is a right to perform a particular operation on an object. An object is a table, view, sequence, procedure, function or package.

System privilege

A system privilege is a right to perform certain operation in the system. For example, the privilege to create a table is a system privilege.

Object Privileges

User owns the object that he/she creates. Unless otherwise specified, only the owner and DBA can access the object.

But, if user wants to share his object with other users, he has to grant privileges on the object to other users.

The following are the list of object privileges available in Oracle

ALTER, DELETE EXECUTE,INDEX,INSERT,SELECT,UPDATE

These privileges are given on various objects, such as

TABLE, VIEW, SEQUENCE, PROCEDURE,FUNCTION, PACKAGE AND OBJECT TYPE

SYNTAX

```
GRANT <PRIVILEGES> | ALL [(COLUMN1,COLUMN2)] ON <OBJECT>
TO (USER | PUBLIC | ROLE) [ WITH GRANT OPTION];
```

Privileges --- any object privileges

ALL -- to grant all privileges
PUBLIC -- to grant privilege to all the users of the system.

Example

```
GRANT SELECT,UPDATE ON EMP TO X;
```

Note :- X is a user

Now from X user, he can select as well as update the information in EMP table.

```
SELECT * FROM SCOTT.EMP;

UPDATE SCOTT.EMP SET SAL = 11000 WHERE EMPNO = 7521;
```

Restricting privilege to certain columns

```
GRANT UPDATE(SAL) ON EMP TO X;
```

So, X user can modify only sal value in the EMP table.

REVOKE OBJECT PRIVILEGES

To remove the given privileges, we can use REVOKE command

```
REVOKE <PRIVILEGES> ON <OBJECT> FROM <USER>;
```

Using Synonyms

To simplify accessing tables owned by other busers, create a SYNONYM. A synonym is another name(alias) to a table or view. By creating a synonym you can avoid giving the owner name while accessing table of other users.

We can create two types of synonyms

Private Synonyms

Public Synonyms

Any user can create private synonym, where as public synonym is created by only DBA using a keyword called public.

Creating private synonym.

```
CREATE SYNONYM EMPL FOR SCOTT.EMP;
```

Now we can access EMP table which is there in SCOTT user using synonym EMPL as

```
SQL> SELECT * FROM EMPL;
```

Public Synonym

Public synonym is available for all the users.

```
SQL>CREATE PUBLIC SYNONYM EMPL FOR SCOTT.EMP;
```

What is SQL*Loader?

SQL*Loader is Oracle's utility program for loading data into an Oracle table.

Most often, SQL*Loader takes two input files – a control file and a data file – and loads the data into a single Oracle table. The data file contains data, and the control file contains information about the data -- where to load it, what to do if something goes wrong, etc.

SQL*Loader has lots and lots of options which can be used to handle various types of data and levels of complexity. SQL*Loader is fully described in the Oracle Server Utilities User's Guide. This document is just about getting started. SQL*Loader runs on Unix, mainframes, and PC's. This document is just about running it from a Windows PC.

Why Use SQL*Loader From Your PC?

If you need to transfer quite a lot of data from your machine to an Oracle database table, you might want to use SQL*Loader. If you already have the data in some other format, it may be worthwhile to use SQL*Loader. If you need to transfer local data to a remote database on some recurring basis, it may be preferable to use SQL*Loader rather than something like FTP. At the end of this document, there is a brief comparison of FTP versus SQL*Loader.

Getting Started, an Example

Say, for example, that you've got an Excel spreadsheet with State data already in it. You've got 50 rows of data – each containing the State Abbreviation, State Name, an [optional] unofficial State Slogan, and the number of State Residents Who Drink Bottled Water.

(Is 50 rows of data really sufficient to justify this exercise? That's debatable, but let's say you've thought it over and you DO want to SQL*load your data into a 4-column Oracle table at UW-Stevens Point. The remote table is called sp.mystates.)

Here's what you do:

1) Create your data file. This is easy. Save your Excel spreadsheet data AS a Comma-Separated-Variable (*.csv) file. This will automatically put commas between each of the four data elements. In addition, if any of the data elements already contain a comma, the Save AS *.csv step will optionally and automatically enclose that data in double quotes.

So, after your Save AS command, you might have a file named `C:\MyStates.csv` that contains data like this:

```
AR,Arkansas,We are sure proud of Bill,0
CO,Colorado,,3000
```

```
WI,Wisconsin,Rose Bowl Champions Again!,5
CA,California,"Dude? You want, like, another hit of Oxygen?",90203049
```

2) Create your control file. Using any text editor, create a file (say, C:\mystates.ctl) containing these lines:

```
LOAD DATA
INFILE 'C:\EMPLOYEE.csv'
REPLACE
INTO TABLE EMPL
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(EMPNO,
 ENAME
INTEGER EXTERNAL)
```

The `REPLACE` keyword says, "remove any existing rows before starting the load." There's also an `INSERT [into empty table]` and `APPEND [to existing rows]` option.

`State_Abbrev, State_Name, State_Slogan, and Nbr_Residents_WDBW` are the actual column names defined in the sp.mystates table.

Because the first three items are of character datatype, it was not necessary to further describe them – character is the default. The fourth column is numeric data – it totals the number of state residents who drink bottled water. The INTEGER EXTERNAL describes the datatype in the C:\mystates.csv input file.

Notice there is some missing data in the data file -- Colorado has no state slogan. The TRAILING NULLCOLS statement handles the missing data; it tells SQL*Loader to load any missing data as NULL values. There are, as we said earlier, *lots* of available options described in the Utilities User's Guide.

3) Create a table with the name referred in ctl file with required columns

4) Run SQL*Loader.

Prerequisites:

- You must have SQL*Loader and SQL*Net installed on your machine. The SQL*Loader program may have a version number included as part of its name, something like `sqlldr73.exe` or `sqlldr80.exe`. Or maybe it will be just `sqlldr.exe`. You can look for it in your ORAWIN95 or ORANT \BIN directory. If it's not installed, you can get the Oracle Client Software installation CD and install "`UTILITIES`".

- You must have the target database (say, it's called 'ELTP') configured as SQL*Net service in your local tnsnames.ora file. This is pretty standard stuff; it's probably already there.

- You must have authorization to modify the sp.mystates table (INSERT, or DELETE and INSERT if you're using the REPLACE option in the control file. In the example below we assume that user SCOTT with password TIGER has appropriate authorization.

At an MS-DOS prompt (or the Start, Run menu) , execute SQL*Loader as follows:

```
sqlldr scott/tiger@ELTP control=C:\mystates.ctl
```

When the load completes, look in the file C:\mystates.log. This log file will contain information about how many rows were loaded, how many rows -- if any -- were NOT loaded, and other information that may be useful to reassure or debug.

How to convert Excel Sheet into CSV(Comma separated variable) file

This article describes how to convert a single column of addresses in a Microsoft Excel worksheet into a comma-separated value (CSV) file that you can import into another program (for example, Microsoft Word).

Note For the address example in this article, the Excel worksheet contains the following address information:

A	B
1	ravi
2	kris
3	babu

1. On the **File** menu, click **Save As**.

   Note In Excel 2007, click the **Microsoft Office Button**, and then click **Save As**.
2. In the **Save As** dialog box:
   a. In the **Save as type** box, click **CSV (Comma delimited) (*.csv)**.
   b. In the **File name** box, type a name for your CSV file (for example, **Address.csv**), and then click **Save**.
   c. Click **OK** when you receive the following message:
   The selected file type does not support workbooks that contain multiple sheets.

   • To save only the active sheet, click OK.
   • To save all sheets, save them individually using a different file name for each, or select a file type that supports multiple sheets.
   d. Click **Yes** when you receive the following message:
   Address.csv may contain features that are incompatible with CSV (comma delimited). Do you want to keep the workbook in this format?

   • To keep this format, which leaves out any incompatible features, click Yes.
   • To preserve the features, click No. Then save a copy in the latest Excel format.
   • To see what might be lost, click Help.
3. On the **File** menu, click **Close**, and then exit Microsoft Excel.

   Note In Excel 2007, click the **Microsoft Office Button**, click **Close**, and then click **Exit Excel**.

   Note You may be prompted to save the file again. When you are prompted, you can click **Yes**, repeat steps c and d, and then exit Excel.

Edit the CSV File in Microsoft Word

1. Start Microsoft Word.
2. On the **File** menu, click **Open**.

   Note In Word 2007, click the **Microsoft Office Button**, and then click **Open**.
3. In the **Files of type** box, click **All Files (*.*)**.
4. Click the CSV file that you saved in step 4 of the "Edit the Excel Worksheet" section, and then click **Open**.
5. On the **Tools** menu, click **Options**.

   Note In Word 2007, skip this step.
6. On the **View** tab, click to select the **All** check box, and then click **OK**.

   Note In Word 2007, follow these steps:
   a. Click the **Microsoft Office Button**, and then click **Word Options**.
   b. Click **Display**.
   c. Click **Paragraph marks** under the **Always show these formatting marks on the screen**.

ANNEXURE

Table : Studies

NAME	NULL	TYPE	
(PNAME)	NOT NULL	VARCHAR2(20)	NAME
SPLACE	NOT NULL	VARCHAR2(20)	STUDIED PLACE
COURSE	NOT NULL	VARCHAR2(20)	COURSE STUDIED

TABLE : SOFTWARE

NAME	NULL ?	TYPE	
(PNAME)	NOT NULL	VARCHAR2(20)	NAME
TITLE	NOT NULL	VARCHAR2(20)	DEVELOPED PROJECT NAME
DEV_IN	NOT NULL	VARCHAR2(10)	LANGUAGE DEVELOPED
SCOST		NUMBER(7,2)	SOFTWARE COST
DCOST		NUMBER(7,2)	DEVELOPMENT COST
SOLD		NUMBER(4)	NO OF SOFTWARE SOLD

Data in Table : STUDIES

PNAME	SPLACE	COURSE	COST
ANAND	SABHARI	PGDCA	45000
ALTAF	COIT	DCA	7200
JULIANA	BITS	MCA	22000
KAMALA	PRAGATHI	DCP	5000
MARY	SABHARI	PGDCA	4600
NELSON	PRAGATHI	DAP	6200
PATRICK	SABHARI	DCA	5200
QADIR	APPLE	HDCP	14000
RAMESH	SABHARI	PGDCA	4500
REBECCA	BPILLANI	DCA	11000
REMITHA	BDPS	DCS	6000
REVATHI	SABHARI	DAP	5000
VIJAYA	BDPS	DCA	48000

TABLE : SOFTWARE

PNAME	TITLE	DEV_IN	SCOST	DCOST	SOLD
ANAND	PARACHUTES	BASIC	399	6000	43
ANAND	VIDEO TITLING PACK	PASCAL	7500	16000	9
JULIANA	INVENTORY CONTROL	COBOL	3000	3500	0
KAMALA	PAYROLL PACKAGE	DBASE	9000	20000	7
MARY	FINANCIAL ACC.S/W	ORACLE	18000	85000	4
PATRICK	CODE GENERATION	COBOL	4500	20000	23
QADIR	READ ME	C++	300	1200	84
QADIR	BOMBS AWAY	ASSEMBLY	750	5000	11
QADIR	VACCINES	C	1900	3400	21
RAMESH	HOTEL MANAGEMENT	DBASE	12000	3500	4
RAMESH	DEAD LEE	PASCAL	599	4500	73
REMITHA	PC UTILITIES	C	725	5000	51
REMITHA	TSR HELP PACKAGE	ASSEMBLY	2500	6000	6
REVATHI	HOSPITAL MANAGEMENT	PASCAL	1100	75000	2
REVATHI	QUIZ MASTER	BASIC	3200	2100	15
VIJAYA	ISR EDITION	C	900	700	6

Data in Table : PROGRAMMER

PNAME	DOB	DOJ	SEX	PROF1	PROF2	SALARY
ANAND	21-APR-66	21-APR-92	M	PASCAL	BASIC	3200
ALTAF	02-JUL-64	13-NOV-90	M	CLIPPER	COBOL	2800
JULIANA	31-JAN-68	21-APR-90	F	COBOL	DBASE	3000
KAMALA	30-OCT-68	02-JAN-92	F	C	DBASE	2900
MARY	24-JUN-70	01-FEB-91	F	C++	ORACLE	4500
NELSON	11-SEP-85	11-OCT-89	M	COBOL	DBASE	2500
PATRICK	10-NOV-65	21-APR-90	M	PASCAL	CLIPPER	2800
QADIR	31-AUG-65	21-APR-91	M	ASSEMBLY	C	3000
RAMESH	03-MAY-67	28-FEB-91	M	PASCAL	DBASE	3200
REBECCA	01-JAN-67	01-DEC-90	F	BASIC	COBOL	2500
REMITHA	19-APR-70	20-APR-93	F	C	ASSEMBLY	3600
REVATHI	02-DEC-69	02-JAN-92	F	PASCAL	BASIC	3700
VIJAYA	14-DEC-65	02-MAY-92	F	FOXPRO	C	3500

QUERY – I

- Find out the selling cost average for packages developed in pascal.
- Display the names and ages of all the programmers
- Display the names of those who have done the DAP course
- What is the highest number of copies sold by a package
- Display the names and date of birth of all programmers born in January
- Display the lowest course fee
- How many programmers have done the PGDCA course
- How much revenue has been earned through sale of packages developed in C
- Display the software's developed by Ramesh
- How many programmers studied at Sabhari?
- Display the details of the packages whose sales crossed 2000 mark.
- Find out the number of copies which should be sold in order to recover the development cost of each package.
- Display the details of packages for which development cost have been recovered.
- What is the price of the costliest software developed in BASIC
- How many packages were developed in DBASE?
- How many programmers studied at Pragathi?
- How many programmers paid 5000 to 10000 for their course?
- What is the average course fee?
- Display the details of programmers knowing C
- How many programmers know either COBOL or PASCAL?
- How many programmers don't know PASCAL and C?
- How old is the oldest male programmer?
- What is the average age of female programmers?
- Calculate the experience in years for each programmer and display along with the names, in descending order.
- Who are the programmers who celebrate their birthday's during the current month?
- How many female programmers are there?
- What are the languages known by the male programmers
- What is the average salary
- How many people draw 2000 to 4000?
- Display the details of those who don't know Clipper, COBOL or Pascal
- Display the details of those who will be completing 2 years of services this year?
- Calculate the amount to be recovered for those packages whose development cost has not yet been recovered?
- List the packages, which have not been sold so far?
- Find out the cost of the software developed by Mary?
- Display the institute names from the studies table without the duplicates

- How many different courses are mentioned in the studies table?
- Display the names of the programmers whose names contain 2 occurrences of the letter ‘A’
- Display the names of the programmers whose names contains 5 characters
- How many female programmers knowing COBOL have more than 2 yrs experience
- What is the length of the shortest name in the programmer table
- What is the average development cost of a package developed in COBOL
- Display the name,sex, dob(dd/mm/yy format) , DOJ(dd/mm/yy format) for all the programmers
- What is the amount paid in salaries of the male programmers who don’t know COBOL
- Display the title , scost, dcost and difference between scost and dcost in descending order of difference
- Display the names of the packages whose names contain more than 1 word
- Display the name, dob, doj of those month of birth and month of joining are same

QUERY – II

- Display the cost of the package developed by each programmer
- Display the sales values of the packages developed by the each programmer
- Display the number of packages sold by each programmer
- Display the sales cost of the packages developed by each programmer
- Display each language name with average development cost, average selling cost and average price per copy

- Display each programmer’s name, costliest package and cheapest packages developed by him / her.
- Display each institute name with number of courses , average cost per course
- Display each Institute name with number of students
- Display the names of male and female programmers
- Display the programmer’s name and their packages
- Display the number of packages in each language except C and C++
- Display the number of packages in each language for which development cost is less than 1000
- Display the average difference between SCOST and DCOST for each language
- Display the total SCOST, DCOST and amount to be recovered for each programmer for those whose dcost has not yet been recovered
- Display the highest, lowest and average salaries for those earning more than 2000

QUERY – III

- Who is the most experienced programmer knowing pascal?
- Which course has been done by the most of the students?
- Which course has the lowest selling cost?
- Display the courses whose fees are within 1000/- (+ or -) of the average fee
- Who developed the package that has sold the least number of copies
- Which language was used to develop most number of packages
- Display the names of the packages which have been sold less than the average number of copies
- Who is the youngest male programmer born in 1965
- In which year most number of programmers born
- Which female programmer earning more than 3000/- does not know C,C++, ORACLE OR DBASE
- Which programmer has developed highest no of packages
- Who are the male programmers earning below the average salary of female programmers
- Display the details of those who are drawing salary
- Display the details of the software developed by the male programmers earning more than 3000
- Display the details of the software developed in C by female programmers of Pragathi
- How many months it will take for each programmer to recover their cost of study
- Display the details of the software developed in the language which is not the programmers first proficiency
- Display the details of the software developed in the language which is neither the first nor second proficiency of the programmer
- Who are the programmers who joined in the same day
- Display the programmers who are drawing same salary
- Which is the costliest package developed by a person with under 3 years experience
- What is the average salary for those whose software sales value is more than 50,000
- How many packages were developed by the person who developed the cheapest package . Where did he / she study.
- How many packages developed by female programmers earning more than the highest paid male programmer
- How many packages were developed by the most experienced programmer from BDPS

QUERY - IV

- Write a query to display the last name, department number, and salary of any employee whose department number and salary both match the department number and salary of any employee who earns a commission.

- Display the last name, department name, and salary of any employee whose salary and commission match the salary and commission of any employee located in location ID 1700.

- Create a query to display the last name, hire date, and salary for all employees who have the same salary and commission as Kochhar.
  Note: Do not display Kochhar in the result set.

- Create a query to display the employees who earn a salary that is higher than the salary of all of the sales managers (JOB_ID = 'SA_MAN'). Sort the results on salary from highest to lowest.

- Find all employees who are not supervisors.
  a. First do this using the NOT EXISTS operator
  b. Can this be done by using the NOT IN operator? How, or why not?

- Write a query to display the last names of the employees who earn less than the average salary in their departments.

- Write a query to display the last names of the employees who have one or more coworkers in their departments with later hire dates but higher salaries.

- Write a query to display the department names of those departments whose total salary cost is above one eighth (1/8) of the total salary cost of the whole company

- Write a query to display the last name, department number, and department name for all employees

- Create a unique listing of all jobs that are in department 80. Include the location of the department in the output.

- Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission.

- Display the employee last name and department name for all employees who have an *a* (lowercase) in their last names

- Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

- display all employees including King, who has no manager. Order the results by the employee number.

- Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label.

- Show the structure of the JOB_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees.

- Create a query to display the name and hire date of any employee hired after employee Davies.

- Write a query to display the following for those employees whose manager ID is less than 120:
  - Manager ID
  - Job ID and total salary for every job ID for employees who report to the same manager
  - Total salary of those managers
  - Total salary of those managers, irrespective of the job IDs

Question: 1

Which SELECT statement should you use if you want to display unique combinations of the POSITION and MANAGER values from the EMPLOYEE table?

1. SELECT DISTINCT position, manager FROM employee;
2. SELECT position, manager DISTINCT FROM employee;
3. SELECT position, manager FROM employee;
4. SELECT position, DISTINCT manager FROM employee;

Question: 2

You need to produce a report for mailing labels for all customers. The mailing label must have only the customer name and address. The CUSTOMERS table has these columns:

CUST_ID	NUMBER(4)	NOT NULL
CUST_NAME	VARCHAR2(100)	
CUST_ADDRESS	VARCHAR2(150)	
CUST_PHONE	VARCHAR2(20)	

Which SELECT statement accomplishes this task?

1. SELECT* FROM customers;
2. SELECT name, address FROM customers;
3. SELECT id, name, address, phone FROM customers;
4. SELECT cust_name, cust_address FROM customers;
5. SELECT cust_id, cust_name, cust_address, cust_phone FROM customers;.

Question: 3

Evaluate this SQL statement:
SELECT e.EMPLOYEE_ID,e.LAST_NAME,e.DEPARTMENT_ID, d.DEPARTMENT_NAME.
FROM EMP e, DEPARTMENT d
WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID;
In the statement, which capabilities of a SELECT statement are performed?

1. Selection, projection, join
2. Difference, projection, join
3. Selection, intersection, join
4. Intersection, projection, join
5. Difference, projection, product

Question: 4

Which two statements are true regarding the ORDER BY clause?

1. The sort is in ascending by order by default.
2. The sort is in descending order by default.
3. The ORDER BY clause must precede the WHERE clause.
4. The ORDER BY clause is executed on the client side.
5. The ORDER BY clause comes last in the SELECT statement.
6. The ORDER BY clause is executed first in the query execution.

Question: 5

From SQL*Plus, you issue this SELECT statement:
SELECT * From orders;

You use this statement to retrieve data from a data table for __________.
1. Updating
2. Viewing
3. Deleting
4. Inserting
5. Truncating

Question: 6

Which SQL SELECT statement performs a projection, a selection, and join when executed?

1. SELECT p.id_number, m.manufacturer_id, m.city
FROM product p, manufacturer m
WHERE p.manufacturer_id = m.manufacturer_id AND m.manufacturer_id = 'NF10032';

2. SELECT id_number, manufacturer_id FROM product
ORDER BY manufacturer_id, id_number;

3. SELECT id_number, manufacturer_id
FROM product
WHERE manufacturer_id = 'NF10032';

4. SELECT manufacturer_id, city
FROM manufacturer
AND manufacturer_id = 'NF10032'
ORDER BY city;

Question: 7

The CUSTOMERS table has these columns:

CUSTOMER_ID	NUMBER(4)	NOT NULL
CUSTOMER_NAME	VARCHAR2(100)	NOT NULL
STREET_ADDRESS	VARCHAR2(150)	
CITY_ADDRESS	VARCHAR2(50)	
STATE_ADDRESS	VARCHAR2(50)	
PROVINCE_ADDRESS	VARCHAR2(50)	
COUNTRY_ADDRESS	VARCHAR2(50)	
POSTAL_CODE	VARCHAR2(12)	
CUSTOMER_PHONE	VARCHAR2(20)	

Which statement finds the rows in the CUSTOMERS table that do not have a postal code?

```
1. SELECT customer_id, customer_name
FROM customers
WHERE postal_code CONTAINS NULL;
```

```
2. SELECT customer_id, customer_name
FROM customers
WHERE postal_code = '________';
```

```
3. SELECT customer_id, customer_name
FROM customers
WHERE postal_code IS NULL;
```

```
4. SELECT customer_id, customer_name
FROM customers
WHERE postal code IS NVL;
```

```
5. SELECT customer_id, customer_name
FROM customers
WHERE postal_code = NULL;
```

Question: 8

Evaluate these two SQL statements:

```
SELECT last_name, salary , hire_date
FROM EMPLOYEES
ORDER BY salary DESC;
```

```
SELECT last_name, salary, hire_date
FROM EMPLOYEES
ORDER BY 2 DESC;
```

What is true about them?

1. The two statements produce identical results.
2. The second statement returns a syntax error.
3. There is no need to specify DESC because the results are sorted in descending order by default.
4. The two statements can be made to produce identical results by adding a column alias for the salary column in the second SQL statement.

Question: 9

Evaluate the set of SQL statements:

```
CREATE TABLE dept
(deptno NUMBER(2),
dname VARCNAR2(14),
loc VARCNAR2(13));
ROLLBACK;
DESCRIBE DEPT
```

What is true about the set?

1. The DESCRIBE DEPT statement displays the structure of the DEPT table.
2. The ROLLBACK statement frees the storage space occupies by the DEPT table.
3. The DESCRIBE DEPT statement returns an error ORA-04043: object DEPT does not exist.
4. The DESCRIBE DEPT statement displays the structure of the DEPT table only if the COMMIT statement introduced before the ROLLBACK statement..

Question: 10

Examine the data of the EMPLOYEES table.

EMPLOYEES (EMPLOYEE_ID is the primary key. MGR_ID is the ID of managers and refers to the EMPLOYEE_ID)

EMPLOYEE_ID	EMP_NAME	DEPT_ID	MGR_ID	JOB_ID	SALARY
101	Smith	20	120	SA_REP	4000
102	Martin	10	105	CLERK	2500
103	Chris	20	120	IT_ADMIN	4200
104	John	30	108	HR_CLERK	2500
105	Diana	30	108	HR_MGR	5000
106	Bryan	40	110	AD_ASST	3000
108	Jennifer	30	110	HR_DIR	6500
110	Bob	40		EX_DIR	8000
120	Ravi	20	110	SA_DIR	6500

Evaluate this SQL statement:

```
SELECT e.employee_id "Emp_id", e.emp_name "Employee",
e.salary, m.employee_id "Mgr_id", m.emp_name "Manager"
FROM employees e, employees m
WHERE e.mgr_id = m.employee_id AND e.salary > 4000;
```

What is its output?

1.

```
EMP_id EMPLOYEE        SALARY      Mgr_id        Manager
------- ---------- --------- ------------- --------------
110     Bob             8000                Bob
120     Ravi            6500           110  Ravi
108     Jennifer        6500           110  Jennifer
103     Chris           4200           120  Chris
105     Diana           5000           108  Diana
```

2.

```
EMP_id        EMPLOYEE  SALARY      Mgr_id        Manager
------- ---------- --------- ------------- --------------
120     Ravi            6500           110  Bob
108     Jennifer        6500           110  Bob
103     Chris           4200           120  Ravi
105     Diana           500            108  Jennifer
```

3.

```
EMP_id        EMPLOYEE  SALARY      Mgr_id        Manager
------- ---------- --------- ------------- --------------
110     Bob             800
120     Ravi            6500           110  Bob
108     Jennifer        6500           110  Bob
103     Chris           4200           120  Ravi
105     Diana           5000           108  Jennifer
```

D

```
EMP_id        EMPLOYEE  SALARY      Mgr_id        Manager
------- ---------- --------- ------------- --------------
110     Bob             8000           110  Bob
120     Ravi            6500           120  Ravi
108     Jennifer        6500           108  Jennifer
103     Chris           4200           103  Chris
105     Diana           5000           105  Dina
```

5. The SQL statement produces an error.

Question: 11

Which /SQL*Plus feature can be used to replace values in the WHERE clause?
1. Substitution variables
2. Replacement variables
3. Prompt variables
4. Instead-of variables
5. This feature cannot be implemented through /SQL*Plus.

Question: 12

You are formulating queries in a SQL*Plus. Which of the following statement correctly describes
how to specify a column alias?
1. Place the alias at the beginning of the statement to describe the table.
2. Place the alias after each column separated by a space to describe the column.
3. Place the alias after each column separated by a comma to describe the column.
4. Place the alias at the end of the statement to describe the table.

Question: 13

You want to use a function in you column clause of a SQL statement. The NVL function
accomplishes which of the following tasks?

1. Assists in the distribution of output across multiple columns.
2. Enables you to specify alternate output for non-NULL column values.
3. Enables you to specify alternated out for NULL column values.
4. Nullifies the value of the column out put.

Question: 14

You want to use SQL*Plus to connect to the oracle database. Which of the following choices
does not indicate a component you must specify when logging into the oracle?

1. The SQL*Plus Keyword.
2. The username
3. The password.
4. The database name.

Question: 15

The EMPLOYEE_HISTORY table contains these columns:
EMPLOYEE_ID NUMBER

LAST_NAME	VARCHAR2(25)
FIRST_NAME	VARCHAR2(25)
DEPARTMENT_ID	NUMBER
POSITION	VARCHAR2(30)
SALARY	NUMBER(6,2)
HIRE_DATE	DATE
DEPART_DATE	DATE

The EMPLOYEE_HISTORY table contains only former employees.
You need to create a report to display all former employees that were hired on or
after January 1, 1996. The data should display in this format:

```
Former Employee    Term of Employment
--------------------------- ---------------------------------
14837 - SMITH               10-MAY-92 / 01-JUN-01
```

Which SELECT statement could you use?

1. SELECT employee_id||' - '||last_name AS Former Employee,
hire_date||' / '||depart_date AS Term of Employment
FROM employee_history
WHERE hire_date > '31-DEC-95';

2. SELECT employee_id||' - '||last_name "AS Former Employee",
hire_date||' / '||depart_date "AS Term of Employment"
FROM employee_history
WHERE hire_date > '31-DEC-95';

3. SELECT employee_id||' - '||last_name 'Former Employee',
hire_date||' / '||depart_date 'Term of Employment'
FROM employee_history
WHERE hire_date > '31-DEC-95' AND depart_date > NULL;

4. SELECT employee_id||' - '||last_name "Former Employee",
hire_date||' / '||depart_date "Term of Employment"
FROM employee_history
WHERE hire_date > '31-DEC-95' AND depart_date <> NULL;

5. SELECT employee_id||' - '||last_name "Former Employee",
hire_date||' / '||depart_date "Term of Employment"
FROM employee_history
WHERE hire_date > '31-DEC-95' AND depart_date IS NOT NULL;

Question: 16

The EMPloyee table contains these columns:

Empno	Number(4)
Ename	Varchar2(10)
job	Varchar2(10)
sal	Varchar2(10)

You need to display the employees information by using this query.
How many columns are presented after executing this query:

SELECT Empno||','||Ename||','||Job "Employee Information" FROM employee;

A) 1
B) 2
C) 3
D) 0
E) 4

Question: 17

Examine the data of the EMPLOYEES table.

EMPLOYEES (EMPLOYEE_ID is the primary key. MGR_ID is the ID of managers and refers to
the EMPLOYEE_ID)

EMPLOYEE_ID	EMP_NAME	DEPT_ID	MGR_ID	JOB_ID	SALARY
101	Smith	20	120	SA_REP	4000
102	Martin	10	105	CLERK	2500
103	Chris	20	120	IT_ADMIN	4200
104	John	30	108	HR_CLERK	2500
105	Diana	30	108	HR_MGR	5000
106	Bryan	40	110	AD_ASST	3000
108	Jennifer	30	110	HR_DIR	6500
110	Bob	40		EX_DIR	8000
120	Ravi	20	110	SA_DIR	6500

Which statement lists the ID, name, and salary of the employee, and the ID and name of the employee's manager, for all the employees who have a manager and earn more than 4000?

1. SELECT employee_id "Emp_id", emp_name "Employee",
salary, employee_id "Mgr_id", emp_name "Manager"
FROM employees
WHERE salary > 4000;

2. SELECT e.employee_id "Emp_id", e.emp_name "Employee",
e.salary, m.employee_id "Mgr_id", m.emp_name "Manager"
FROM employees e, employees m
WHERE e.mgr_id = m.mgr_id AND e.salary > 4000;

3. SELECT e.employee_id "Emp_id", e.emp_name "Employee",
e.salary, m.employee_id "Mgr_id", m.emp_name "Manager"
FROM employees e, employees m
WHERE e.mgr_id = m.employee_id AND e.salary > 4000;

4. SELECT e.employee_id "Emp_id", e.emp_name "Employee",
e.salary, m.mgr_id "Mgr_id", m.emp_name "manager"
FROM employees e, employees m
WHERE e.mgr_id = m.employee_id AND e.salary > 4000;

5. SELECT e.employee_id "Emp_id", e.emp_name "Employee",

```
e.salary, m.mgr_id "Mgr_id", m.emp_name "Manager"
FROM employees e, employees m
WHERE e.employee_id = m.employee_id
AND e.salary > 4000;.
```

Question: 18

You need to display the last names of those employees who have the letter “A” as the second
character in their names.
Which SQL statement displays the required results?
1. SELECT last_name FROM EMP WHERE last_name LIKE ‘_A%’;
2. SELECT last_name FROM EMP WHERE last name =’*A%’
3. SELECT last_name FROM EMP WHERE last name =’_A%’;
4. SELECT last_name FROM EMP WHERE last name LIKE ‘*A%’

Question: 19

In which scenario would TOP N analysis be the best solution?
1. You want to identify the most senior employee in the company.
2. You want to find the manager supervising the largest number of employees.
3. You want to identify the person who makes the highest salary for all employees.
4. You want to rank the top three sales representatives who have sold the maximum number of
products.

Question: 20

Evaluate this SQL statement:

```
SELECT c.customer_id, o.order_id, o.order_date, p.product_name
FROM customer c, curr_order o, product p
WHERE customer.customer_id = curr_order.customer_id AND o.product_id
= p.product_id
ORDER BY o.order_amount;
```

This statement fails when executed. Which change will correct the problem?
1. Include the ORDER_AMOUNT column in the SELECT list.
2. Use the table name in the ORDER BY clause.
3. Remove the table aliases from the WHERE clause.
4. Use the table aliases instead of the table names in the WHERE clause.
5. Remove the table alias from the ORDER BY clause and use only the column name.

Question: 21

You want to display the titles of books that meet these criteria:
1. Purchased before January 21, 2001
2. Price is less then $500 or greater than $900

You want to sort the results by their data of purchase, starting with the most recently bought
book.
Which statement should you use?

```
1. SELECT book_title
FROM books
WHERE price between 500 and 900 AND purchase_date < ’21-JAN-2001’
ORDER BY purchase_date;
```

```
2. SELECT book_title
FROM books
WHERE price IN (500,900) AND purchase_date < ’21-JAN-2001’
ORDER BY purchase date ASC;
3. SELECT book_title
FROM books
WHERE price < 500 or > 900 AND purchase_date < ’21-JAN-2001’
ORDER BY purchase date DESC;
```

```
4. SELECT book_title
FROM books
WHERE (price < 500 OR price > 900) AND purchase_date < ’21-JAN-2001’
ORDER BY purchase date DESC;
```

Question: 22

For which task would you use the WHERE clause in a SELECT statement?
1. to designate the ORDER table location
2. to compare PRODUCT_ID values to 7382
3. to display only unique PRODUCT_ID values
4. to restrict the rows returned by a GROUP BY clause

Question: 23

The STUDENT_GRADES table has these columns:
STUDENT_ID NUMBER(12)
SEMESTER_END DATE
GPA NUMBER(4,3)
The registrar has requested a report listing the students' grade point averages (GPA), sorted from
highest grade point average to lowest within each semester, starting from the earliest date. Which
statement accomplishes this?

```
1. SELECT student_id, semester_end, gpa
FROM student_grades
ORDER BY semester_end DESC, gpa DESC;
```

```
2. SELECT student_id, semester_end, gpa
FROM student_grades
ORDER BY semester_end ASC, gpa ASC;
```

```
3. SELECT student_id, semester_end, gpa
FROM student_grades
ORDER BY semester_end, gpa DESC;
```

```
4. SELECT student_id, semester_end, gpa
FROM student_grades
ORDER BY gpa DESC, semester_end DESC;
```

```
5. SELECT student_id, semester_end, gpa
FROM student_grades
ORDER BY gpa DESC, semester_end ASC;.
```

Question: 24

The ORDERS table has these columns:

ORDER_ID	NUMBER(4)	NOT NULL
CUSTOMER_ID	NUMBER(12)	NOT NULL
ORDER_TOTAL	NUMBER(10,2)	

The ORDERS table tracks the Order nnmher, the order total, and the customer to whom the
Order belongs. Which two statements retrieve orders with an inclusive total that ranges between
100.00 and 2000.00 dollars?

```
1. SELECT customer_id, order_id, order_total FROM orders
RANGE ON order_total (100 AND 2000) INCLUSIVE;
```

```
2. SELECT customer_id, order_id, order_total FROM orders
HAVING order_total BETWEEN 100 and 2000;
```

```
3. SELECT customer_id, order_id, order_total FROM orders
WHERE order_total BETWEEN 100 and 2000;
```

```
4. SELECT customer_id, order_id, order_total FROM orders
WHERE order_total >= 100 and <= 2000;
```

```
5. SELECT customer_id, order_id, order_total FROM orders
WHERE order_total >= 100 and order_total <= 2000;
```

Question: 25

Examine the structure of the PRODUCT table.

PRODUCT Table

PRODUCT _ID	NUMBER Primary Key

PRODUCT_NAME	VARCHAR2(25)
SUPPLIER_ID	NUMBER references SUPPLIER(SUPPLIER_ID)
CATERORY_ID	NUMBER
QTY_PER_UNIT	NUMBER
LIST_RRICE	NUMBER (5,2)
COST	NUMBER (5,2)

You want to display all product identification numbers of products for which there are 500 or more
available for immediate sale. You want the product numbers displayed alphabetically by supplier,
then by product number from lowest to highest. Which statement should you use to achieve the
required results?

```
1. SELECT product_id
FROM product
WHERE qty_per_unit >= 500
ORDER BY supplier_id, product_id;
```

```
2. SELECT product_id
FROM product
WHERE qty_per_unit >= 500
SORT BY supplier_id, product_id;
```

```
3. SELECT product_id
FROM product
WHERE qty_per_unit >= 500
ORDER BY supplier_id, product_id DESC;
```

```
4. SELECT product_id
FROM product
WHERE qty_per_unit > 500
SORT BY supplier_id, product_id;
```

Question: 26

Examine the data in TEACHER table.

ID	LAST_NAME	FIRST_NAME	SUBJECT_ID
88	Tsu	Ming	HST_AMER
70	Smith	Ellen	HST_INDIA
56	Jones	Karen	HST_REVOL
58	Hann	Jeff	HST_CURR
63	Hopewell	Mary Elizabetn	HST_RELIG

Which query should you use to return the following values from the TEACHER table?

```
Name                              Subject
------------------------------------ -------------------
Jones, Karen                              HST_REVOL
Hopewell, Mary Elizabeth  HST_RELIG
```

```
1. SELECT last_name||', '||first_name "Name", subject_id "Subject"
FROM teacher
WHERE subject_id LIKE 'HST\_%' ESCAPE '\';
```

```
2. SELECT last_name||', '||first_name "Name", subject_id "Subject"
FROM teacher
WHERE subject_id = 'HST\_R%';
```

```
3. SELECT last_name||', '||first_name "Name", subject_id "Subject"
FROM teacher
WHERE subject_id LIKE '%HST\_R%' ESC '\ ';
```

```
4. SELECT last_name||', '||first_name "Name", subject_id "Subject"
FROM teacher
WHERE subject_id LIKE 'HST_%';
```

Question: 27

You query the database with this SQL statement:

```
SELECT bonus
FROM salary
WHERE bonus BETWEEN 1 AND 250 OR (bonus IN(190, 500, 600)
AND bonus BETWEEN 250 AND 500);
```

Which value could the statement return?

1. 100
2. 260
3. 400
4. 600

Question: 28

Examine the structure of the STUDENTS table:

STUDENT_ID	NUMBER	Primary Key
STUDENT_NAME	VARCHAR2(30)	
COURSE_ID	VARCHAR2(10)	NOT NULL
MARKS	NUMBER	
START_DATE	DATE	
FINISH_DATE	DATE	

You need to create a report of the 10 students who achieved the highest ranking in the course INT SQL and who completed the course in the year 1999.

Which SQL statement accomplishes this task?

```
1. SELECT student_ id, marks, ROWNUM "Rank"
FROM students
WHERE ROWNUM <= 10
AND finish_date BETWEEN '01-JAN-99' AND '31-DEC-99'
AND course_id = 'INT_SQL'
ORDER BY marks DESC;
```

```
2. SELECT student_id, marks, ROWID "Rank"
FROM students
WHERE ROWID <= 10
AND finish_date BETWEEN '01-JAN-99' AND '31-DEC-99'
AND course_id = 'INT_SQL'
ORDER BY marks;
```

```
3. SELECT student_id, marks, ROWNUM "Rank"
FROM (SELECT student_id, marks
FROM students
WHERE ROWNUM <= 10
AND finish_date BETWEEN '01-JAN-99' AND
'31-DEC-99'
AND course_id = 'INT_SQL'
ORDER BY marks DESC);
```

```
4. SELECT student_id, marks, ROWNUM "Rank”
FROM (SELECT student_id, marks
FROM students
ORDER BY marks DESC)
WHERE ROWNUM <= 10
AND finish_date BETWEEN '01-JAN-99' AND '31-DEC-99'
AND course_id = 'INT_SQL';
```

Question: 29

Examine the structure of the LINE_ITEM table.

```
LINE_ITEM_ID            NUMBER(9),
ORDER_ID           NUMBER(9) NOT NULL,
PRODUCT_ID NUMBER(9) NOT_NULL,
QUANTITY NUMBER(9),
Constraint primary key (LINE_ITEM_ID, ORDER_ID),
Constraint foreign key ORDER_ID REFERENCES
CURR_ORDER(ORDER_ID),
Constraint foreign key PRODUCT_ID REFERENCES
PRODUCT(PRODUCT_ID));
```

You must display the order number, line item number, product identification number, and quantity of each item where the quantity ranges from 10 through 100. The order numbers must be in the range of 1500 through 1575. The results must be sorted by order number from lowest to highest and then further sorted by quantity from highest to lowest.

Which statement should you use to display the desired result?

```
1. SELECT order_id, line_item_id, product_id, quantity
FROM line_item
```

WHERE quantity BETWEEN 9 AND 101 AND order_id BETWEEN 1500 AND 1575
ORDER BY order_id DESC, quantity DESC;

2. SELECT order_id, line_item_id, product_id, quantity
FROM line_item
WHERE (quantity > 10 AND quantity < 100) AND order_id BETWEEN 1500 AND 1575
ORDER BY order_id ASC, quantity;

3. SELECT order_id, line_item_id, product_id, quantity
FROM line_item
WHERE (quantity > 9 OR quantity < 101) AND order_id BETWEEN 1500 AND 1575
ORDER BY order_id, quantity;

4. SELECT order_id, line_item_id, product_id, quantity
FROM line_item
WHERE quantity BETWEEN 10 AND 100 AND order_id BETWEEN 1500 AND 1575
ORDER BY order_id, quantity DESC;

Question: 30

The ITEM table contains these columns:

ITEM_ID	NUMBER(9)
COST	NUMBER(7,2)
RETAIL	NUMBER(7,2)

You need to create a report that displays the cost, the retail price, and the profit for item number
783920. To calculate the profit, subtract the cost of the item from its retail price, and then deduct
an administrative fee of 25 percent of this derived value.

Which SELECT statement produces the desired results?

1. SELECT cost, retail, (retail - cost) - ((retail - cost) * .25) "Profit" FROM item
WHERE item_id = 783920;

2. SELECT cost, retail, (retail - cost) - retail - (cost * .25) "Profit" FROM item
WHERE item_id = 783920;

3. SELECT cost, retail, (retail - cost - retail - cost) * .25 "Profit" FROM item
WHERE item_id = 783920;

4. SELECT cost, retail, retail - cost - retail - cost * .25 "Profit" FROM item
WHERE item_id = 783920;

Question: 31

The ITEM table contains these columns:

ITEM_ID	NUMBER(9)
COST	NUMBER(7,2)
RETAIL	NUMBER(7,2)

The RETAIL and COST columns contain values greater than zero.
Evaluate these two SQL statements:
1. SELECT item_id, (retail * 1.25) + 5.00 - (cost * 1.10) - (cost * .10) AS Calculated Profit
FROM item;
2. SELECT item_id, retail * 1.25 + 5.00 - cost * 1.10 - cost * .10 "Calculated Profit"
FROM item;
What will be the result?
1. Statement 1 will display the 'Calculated Profit' column heading.
2. Statement 1 and statement 2 will return the same value.
3. Statement 1 will return a higher value than statement 2.
4. One of the statements will NOT execute.

Question: 32

The EMP table contains these columns:

LAST NAME	VARCHAR2(25)
SALARY	NUMBER(6,2)
DEPARTMENT_ID	NUMBER(6)

You need to display the employees who have not been assigned to any department.
You write the SELECT statement:
SELECT LAST_NAME, SALARY, DEPARTMENT_ID FROM EMP
WHERE DEPARTMENT_ID = NULL;

What is true about this SQL statement?
1. The SQL statement displays the desired results.
2. The column in the WHERE clause should be changed to display the desired results.
3. The operator in the WHERE clause should be changed to display the desired results.
4. The WHERE clause should be changed to use an outer join to display the desired results.

Question: 33

Which two statements are true about WHERE and HAVING clauses?
1. A WHERE clause can be used to restrict both rows and groups.
2. A WHERE clause can be used to restrict rows only.
3. A HAVING clause can be used to restrict both rows and groups.

4. A HAVING clause can be used to restrict groups only.
5. A WHERE clause CANNOT be used in a query of the query uses a HAVING clause.
6. A HAVING clause CANNOT be used in subqueries.

Question: 34

You are sorting data in a table in you SELECT statement in descending order. The column you are sorting on contains NULL records, where will the NULL record appears?
1. At the beginning of the list.
2. At the end of the list.
3. In the middle of the list.
4. At the same location they are listed in the unordered table.

Question: 35

The ACCOUNT table contains these columns:

ACCOUNT_ID	NUMBER(12)
PREVIOUS_BALANCE	NUMBER(7,2)
PAYMENTS	NUMBER(7,2)
NEW_PURCHASES	NUMBER(7,2)
CREDIT_LIMIT	NUMBER(7)

You need to display the account number, finance charge, and current balance for accounts 1500 through 2000 with a current balance greater than the account's credit limit. The finance charge is .9 percent (.009) of the previous balance. Adding the previous balance value, new purchases value, and finance charge value, and then subtracting the payments value yields the current balance value.

Evaluate this statement:

```
SELECT account_id, NVL(previous_balance, 0) * .009 finance_charge,
NVL(new_purchases, 0) + (NVL(previous_balance, 0) * 1.009) -
NVL(payments, 0)
current balance FROM account WHERE (new_purchases +
(previous_balance * 1.009)) - payments > credit_limit AND account_id
BETWEEN 1500 AND 2000;
```

Which statement about this SELECT statement is true?
1. The statement calculates the finance charge incorrectly.
2. The statement calculates the current balance incorrectly.
3. The statement returns only accounts that have NO previous balance.
4. The statement returns only accounts that have new purchases, previous balance, and payments values.

Question: 36

Examine the description of the EMPLOYEES table:

EMP_ID	NUMBER(4)	NOT NULL
LAST_NAME	VARCHAR2(30)	NOT NULL
FIRST_NAME	VARCHAR2(30)	
DEPT_ID	NUMBER(2)	
JOB_CAT	VARCHARD2(30)	
SALARY	NUMBER(8,2)	

Which statement shows the maximum salary paid in each job category of each department?

```
1. SELECT dept_id, job_cat, MAX(salary)
FROM employees
WHERE salary > MAX(salary);
```

```
2. SELECT dept_id, job_cat, MAX(salary)
FROM employees
GROUP BY dept_id, job_cat;
```

```
3. SELECT dept_id, job_cat, MAX(salary)
FROM employees;
```

```
4. SELECT dept_id, job_cat, MAX(salary)
FROM employees
GROUP BY dept_id;
```

```
5. SELECT dept_id, job_cat, MAX(salary)
FROM employees
GROUP BY dept_id, job_cat, salary;
```

Question: 37

Management has asked you to calculate the value 12*salary* comossion_pct for all the
employees in the EMP table. The EMP table contains these columns:

LAST NAME	VARCNAR2(35) NOT NULL
SALARY	NUMBER(9,2) NOT NULL
COMMISION_PCT	NUMBER(4,2)

Which statement ensures that a value is displayed in the calculated columns for all employees?

```
1. SELECT last_name, 12*salary*commison_pct FROM emp;
2. SELECT last_name, 12*salary* (commission_pct,0) FROM emp;
3. SELECT last_name, 12*salary*(nvl(commission_pct,0)) FROM emp;
4. SELECT last_name, 12*salary*(decode(commission_pct,0)) FROM emp;
```

Question: 38

Examine the description of the STUDENTS table:

STD_ID NUMBER(4)
COURSE_ID VARCHARD2(10)
START_DATE DATE
END_DATE DATE.
Which two aggregate functions are valid on the START_DATE column?
1. SUM(start_date)
2. AVG(start_date)
3. COUNT(start_date)
4. AVG(start_date, end_date)
5. MIN(start_date)
6. MAXIMUM(start_date)

Question: 39

The EMPLOYEE tables has these columns:
LAST_NAME VARCNAR2(35)
SALARY NUMBER(8,2)
COMMISSION_PCT NUMBER(5,2)
You want to display the name and annual salary multiplied by the commission_pct for all
employees. For records that have a NULL commission_pct, a zero must be displayed against the
calculated column. Which SQL statement displays the desired results?

1. SELECT last_name, (salary * 12) * commission_pct FROM EMPLOYEES;
2. SELECT last_name, (salary * 12) * IFNULL(commission_pct,0) FROM EMPLOYEES;
3. SELECT last_name, (salary * 12) * NVL2(commission_pct, 0) FROM EMPLOYEES;
4. SELECT last_name, (salary * 12) * NVL(commission_pct, 0) FROM EMPLOYEES;

Question: 40

You would like to display the system date in the format "Monday, 01 June, 2001".
Which SELECT statement should you use?

1. SELECT TO_DATE(SYSDATE, 'FMDAY, DD Month, YYYY') FROM dual;
2. SELECT TO_CHAR(SYSDATE, 'FMDD, DY Month, 'YYY') FROM dual;
3. SELECT TO_CHAR(SYSDATE, 'FMDay, DD Month, YYYY') FROM dual;
4. SELECT TO_CHAR(SYSDATE, 'FMDY, DDD Month, YYYY')
5. SELECT TO_DATE(SYSDATE, 'FMDY, DDD Month, YYYY') FROM dual;

Question: 41

Evaluate the SQL statement:
SELECT ROUND(TRUNC(MOD(1600,10),-1),2)
FROM dual;

What will be displayed?
1. 0
2. 1
3. 0.00
4. An error statement

Question: 42

Examine the description of the MARKS table:

STD_ID	NUMBER(4)
STUDENT_NAME	VARCHAR2(30)
SUBJ1	NUMBER(3)
SUBJ2	NUMBER(3)

SUBJ1 and SUBJ2 indicate the marks obtained by a student in two subjects.

Examine this SELECT statement based on the MARKS table:

```
SELECT subj1+subj2 total_marks, std_id
FROM marks
WHERE subj1 > AVG(subj1) AND subj2 > AVG(subj2)
ORDER BY total marks;
```

What is the result of the SELECT statement?

1. The statement executes successfully and returns the student ID and sum of all marks for
each student who obtained more than the average mark in each subject.
2. The statement returns an error at the SELECT clause.
3. The statement returns an error at the WHERE clause.
4. The statement returns an error at the ORDER BY clause.

Question: 43

Which three SELECT statements displays 2000 in the format “$2,000.00”?
1. SELECT TO_CHAR (2000, ‘$#,###.##’) FROM dual;
2. SELECT TO_CHAR (2000, ‘$0,000.00’) FROM dual;
3. SELECT TO_CHAR (2000, ‘$9,999.00’) FROM dual;
4. SELECT TO_CHAR (2000, ‘$9,999.99’) FROM dual;
5. SELECT TO_CHAR (2000, ‘$2,000.00’) FROM dual;
6. SELECT TO_CHAR (2000, ‘$N,NNN.NN ’) FROM dual;

Question: 44

Examine the description of the EMPLOYEES table:

EMP_ID	NUMBER(4)	NOT NULL
LAST_NAME	VARCHAR2(30)	NOT NULL
FIRST_NAME	VARCHAR2(30).	
DEPT_ID	NUMBER(2)	
JOB_CAT	VARCHAR2(30)	
SALARY	NUMBER(8,2)	

Which statement shows the department ID, minimum salary, and maximum salary paid in that department, only of the minimum salary is less then 5000 and the maximum salary is more than 15000?

```
1. SELECT dept_id, MIN(salary(, MAX(salary)
FROM employees
WHERE MIN(salary) < 5000 AND MAX(salary) > 15000;
```

```
2. SELECT dept_id, MIN(salary), MAX(salary)
FROM employees
WHERE MIN(salary) < 5000 AND MAX(salary) > 15000
GROUP BY dept_id;
```

```
3. SELECT dept_id, MIN(salary), MAX(salary)
FROM employees
HAVING MIN(salary) < 5000 AND MAX(salary) > 15000;
```

```
4. SELECT dept_id, MIN(salary), MAX(salary)
FROM employees
GROUP BY dept_id
HAVING MIN(salary) < 5000 AND MAX(salary) > 15000;
```

```
5. SELECT dept_id, MIN(salary), MAX(salary)
FROM employees
GROUP BY dept_id, salary
HAVING MIN(salary) < 5000 AND MAX(salary) > 15000;
```

Question: 45

Which two are true about aggregate functions?

1. You can use aggregate functions in any clause of a SELECT statement.
2. You can use aggregate functions only in the column list of the SELECT clause and in the WHERE clause of a SELECT statement.
3. You can mix single row columns with aggregate functions in the column list of a SELECT statement by grouping on the single row columns.
4. You can pass column names, expressions, constants, or functions as parameters to an aggregate function.
5. You can use aggregate functions on a table, only by grouping the whole table as one single group.
6. You cannot group the rows of a table by more than one column while using aggregate functions.

Question: 46

Which four statements correctly describe functions that are available in SQL?

1. INSTR returns the numeric position of a named character.
2. NVL2 returns the first non-null expression in the expression list.
3. TRUNCATE rounds the column, expression, or value to n decimal places.
4. DECODE translates an expression after comparing it to each search value.
5. TRIM trims the heading of trailing characters (or both) from a character string.
6. NVL compares two expressions and returns null if they are equal, or the first expression of
they are not equal.
7. NULLIF compares two expressions and returns null if they are equal, or the first expression if
they are not equal.

Question: 47

Examine the structures of the PATIENT, PHYSICIAN, and ADMISSION tables.

PATIENT Table

PATIENT_ID	NUMBER	Primary Key
LAST_NAME	VARCHAR2 (30)	
FIRST_NAME	VARCHAR2 (25)	
DOB	DATE	
INS_CODE	NUMBER	

PHYSICIAN Table

PHYSICIAN _ID	NUMBER	Primary Key
LAST_NAME	VARCHAR2 (30)	NOT NULL
FIRST_NAME	VARCHAR2 (25)	NOT NULL
LICENSE_NO	NUMBER (7)	NOT NULL
HIRE_DTAE	DATE	

ADMISSION Table
PATIENT_ID NUMBER NOT NULL, Primary Key, References PATIENT_ID column of
the PATIENT table
PHYSICIAN_ID NUMBER NOT NULL, Primary Key, References PHYSICIAN_ID column
of the PHYSICIA table
ADMIT_DATE DATE
DISCHG_DATE DATE
ROOM_ID NUMBER Foreign key to ROOM_ID of the ROOM table

Which SQL statement will produce a list of all patients who have more than one physician?

1. SELECT p.patient_id FROM patient p WHERE p.patient_id IN (SELECT patient_id
   FROM admission GROUP BY patient_id HAVING COUNT(*) > 1);
2. SELECT DISTINCT a.patient_id FROM admission a, admission a2 WHERE a.patient_id = a2.patient_id AND a.physician_id <> a2.physician_id;

3. SELECT patient_id FROM admission WHERE COUNT(physician_id) > 1;

4. SELECT patient_id FROM patient FULL OUTER JOIN physician;

Question: 48

Which clause should you use to exclude group results?
1. WHERE
2. HAVING
3. RESTRICT
4. GROUP BY
5. ORDER BY

Question: 49

In a SELECT statement that includes a WHERE clause, where is the GROUP BY clause placed
in the SELECT statement?
1. Immediately after the SELECT clause
2. Before the WHERE clause
3. Before the FROM clause
4. After the ORDER BY clause
5. After the WHERE clause

Question: 50

Which two are character manipulation functions?
1. TRIM
2. REPLACE
3. TRUNC
4. TO_DATE
5. MOD
6. CASE