

Module n°1

Basic select statement

1Z0-007

Auteur : Reda Benkirane
Basic select statement – 3 September 2004
Nombre de pages : 46



Ecole Supérieure d'Informatique de Paris
23. rue Château Landon 75010 – PARIS
www.supinfo.com

1.1. ORACLE 9i FEATURES.....	4
1.1.1. ORACLE 9i APPLICATION SERVER.....	4
1.1.2. ORACLE 9i DATABASE.....	4
1.2. ORACLE INTERNET PLATFORM.....	5
1.3. DATA STORAGE ON DIFFERENT MEDIA	6
1.4. RELATIONAL DATABASE CONCEPT.....	7
1.4.1. Definition of a Relational Database.....	7
1.4.2. Data Models.....	7
1.4.3. Entity Relationship Model.....	8
1.4.4. Entity Relationship modelling conventions.....	8
1.4.5. Relating multiple tables.....	8
1.5. COMMUNICATING WITH A RDBMS USING SQL.....	9
1.6. SQL STATEMENTS.....	10
2. WRITING BASIC SQL SELECT STATEMENTS.....	11
2.1. SELECT STATEMENT.....	11
2.1.1. Capabilities of SQL SELECT statement.....	11
2.1.2. Basic SELECT statement.....	12
2.1.3. Selecting columns.....	12
2.1.4. Writing SQL statements.....	13
2.1.5. Arithmetic expressions.....	13
2.2. CUSTOMIZED STATEMENTS.....	15
2.2.1. Using column Aliases.....	15
2.2.2. Using concatenation operator.....	16
2.2.3. Using literal character string.....	17
2.2.4. Eliminating duplicate rows.....	17
2.3. SQL AND iSQL*PLUS INTERACTION	17
2.3.1. SQL statements AND iSQL*PLUS commands.....	17
2.3.2. Using iSQL*PLUS.....	18
2.3.3. Displaying table structure using DESCRIBE command.....	19
2.3.4. Interacting with script files on iSQL*PLUS.....	20
2.3.5. Interacting with script files using commands.....	21
2.3.6. SQL*PLUS editing commands.....	21
3. RESTRICTING AND STORING DATA.....	23
3.1. LIMITING ROWS USING A SELECTION.....	23
3.1.1. Using the WHERE clause	23
3.1.2. Using WHERE clause with different data types.....	24
3.2. COMPARISON OPERATORS.....	24
3.2.1. Arithmetic operator.....	24
3.2.2. Using the BETWEEN condition.....	24
3.2.3. Using the LIKE condition.....	25
3.2.4. Using the IN condition.....	26
3.2.5. Using the NULL condition.....	27
3.3. LOGICAL CONDITIONS.....	27
3.3.1. Using the AND operator.....	27
3.3.2. Using the OR operator.....	28
3.3.3. Using the NOT operator.....	28
3.3.4. Rules of precedence.....	29
3.4. SORTING RESULTS.....	30
3.4.1. Using the ORDER BY clause.....	30
3.4.2. Sorting in descending order.....	30
3.4.3. Sorting by column alias.....	30
3.4.4. Sorting by multiple columns.....	31
4. SINGLE-ROW FUNCTIONS.....	32
4.1. SQL FUNCTIONS.....	32
4.2. CHARACTER FUNCTIONS.....	33
4.2.1. Case manipulation functions.....	33
4.2.2. Character-manipulation functions.....	33
4.3. NUMBER FUNCTIONS.....	34

4.3.1. Using the <i>ROUND</i> function.....	34
4.3.2. Using the <i>TRUNC</i> function.....	34
4.3.3. Using the <i>MOD</i> function.....	35
4.4. WORKING WITH DATES.....	35
4.4.1. The <i>SYSDATE</i> function.....	35
4.4.2. Using arithmetic operators with dates.....	35
4.4.3. Dates functions.....	36
4.5. CONVERSION FUNCTIONS.....	37
4.5.1. Implicit data type conversion.....	37
4.5.2. Explicit data type conversion.....	37
4.5.3. Using the <i>TO_CHAR</i> function with dates.....	38
4.5.4. Elements of the date format model.....	38
4.5.5. Using the <i>TO_CHAR</i> function with numbers.....	39
4.5.6. Using the <i>TO_NUMBER</i> and <i>TO_DATE</i> functions.....	40
4.5.7. RR date format.....	41
4.6. NESTING FUNCTIONS.....	42
4.7. GENERAL FUNCTIONS.....	43
4.7.1. Using the <i>NVL</i> function.....	43
4.7.2. Using the <i>NVL2</i> function.....	43
4.7.3. Using the <i>NULLIF</i> function.....	44
4.7.4. Using the <i>COALESCE</i> function.....	44
4.8. CONDITIONAL EXPRESSIONS.....	45
4.8.1. Using the <i>CASE</i> expression.....	45
4.8.2. Using the <i>DECODE</i> function.....	46

1.Introduction

1.1.ORACLE 9i features

There are two products, Oracle9i Application Server and Oracle9i Database that provide a complete and simple infrastructure for Internet applications.

1.1.1.ORACLE 9i APPLICATION SERVER

The Oracle9i Application Server (Oracle9iAS) runs all your applications. The Oracle9i Database stores all your data.

Oracle9iAS is the only application server to include services for all the different server applications you will want to run:

- Portals or Web sites
- Java transactional applications
- Business intelligence applications

It also provides integration between users, applications, and data throughout your organization.

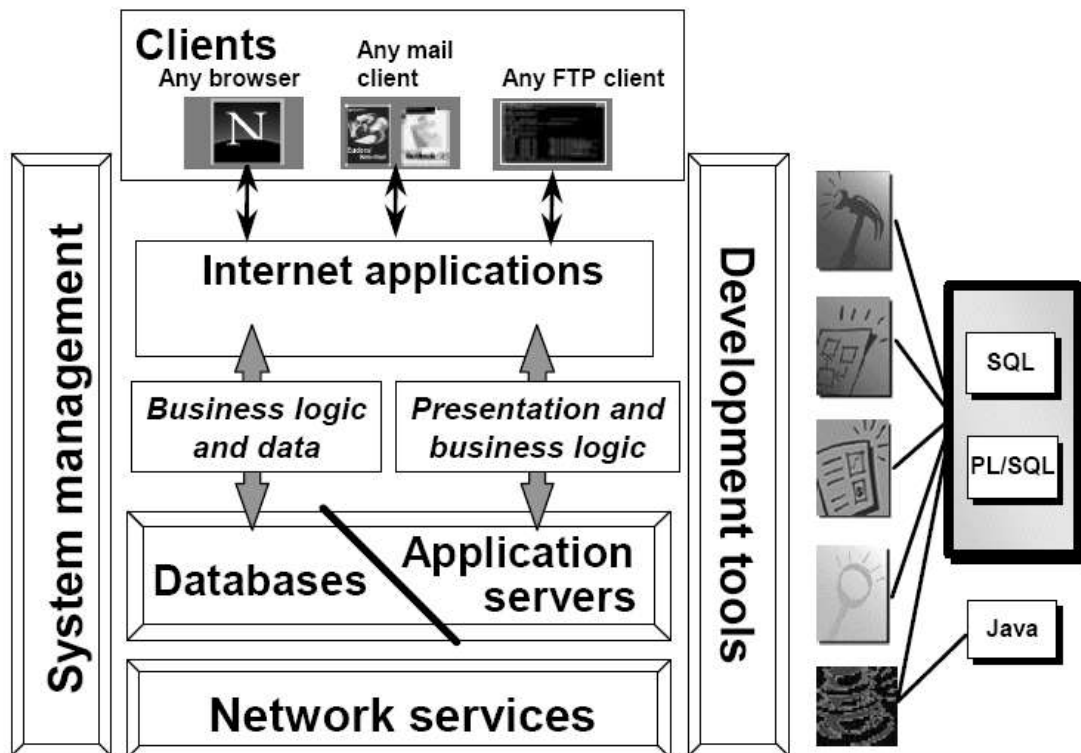
1.1.2.ORACLE 9i DATABASE

Oracle9i Database has services through which you can store metadata about information stored in file systems.

You can use the database server to manage and serve information wherever it is located.

Oracle9i Database is the only database specifically designed as an internet development and deployment platform, extending Oracle's long-standing technology leadership in the areas of data management, transaction processing, and data warehousing to the new medium of the internet.

1.2. Oracle Internet Platform



Oracle offers a high-performance Internet platform that includes everything needed to develop, to deploy, and to manage Internet applications.

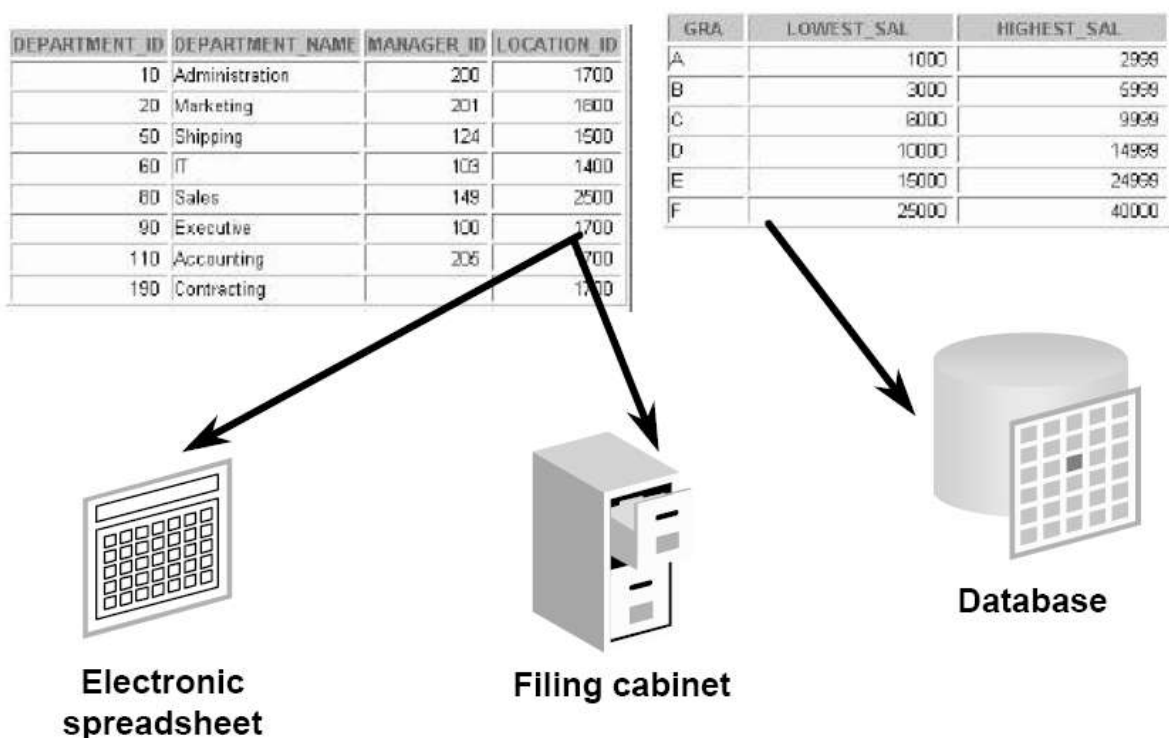
This platform is built on three core pieces:

- Browser-based clients to process presentation
- Application servers to execute business logic and serve presentation logic to browser-based clients
- Databases to execute database-intensive business logic and serve data

Oracle offers many development tools to build business applications.

Stored procedures, functions, and packages can be written by using SQL, PL/SQL, or JAVA.

1.3. Data storage on different media



A company needs to save information about employees, departments, or salaries and more. These information are called data.

Data can be stored on different formats, such as a hard-copy document in a filing cabinet or in databases.

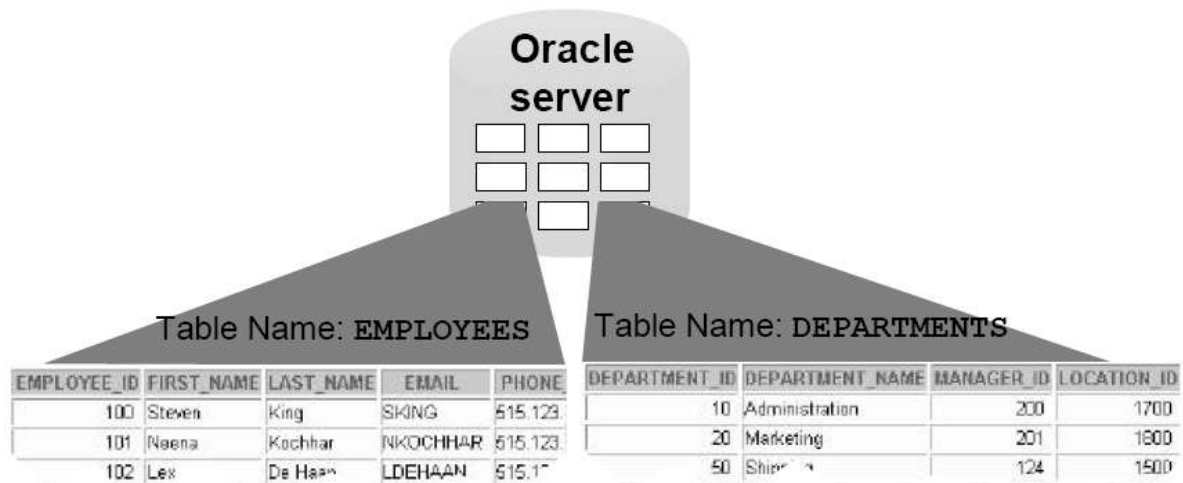
A database is an organized collection of information; there are four main types of databases:

- Hierarchical
- Network
- Relational
- Object relational

Oracle9i is an Object relational database management system (DBMS).

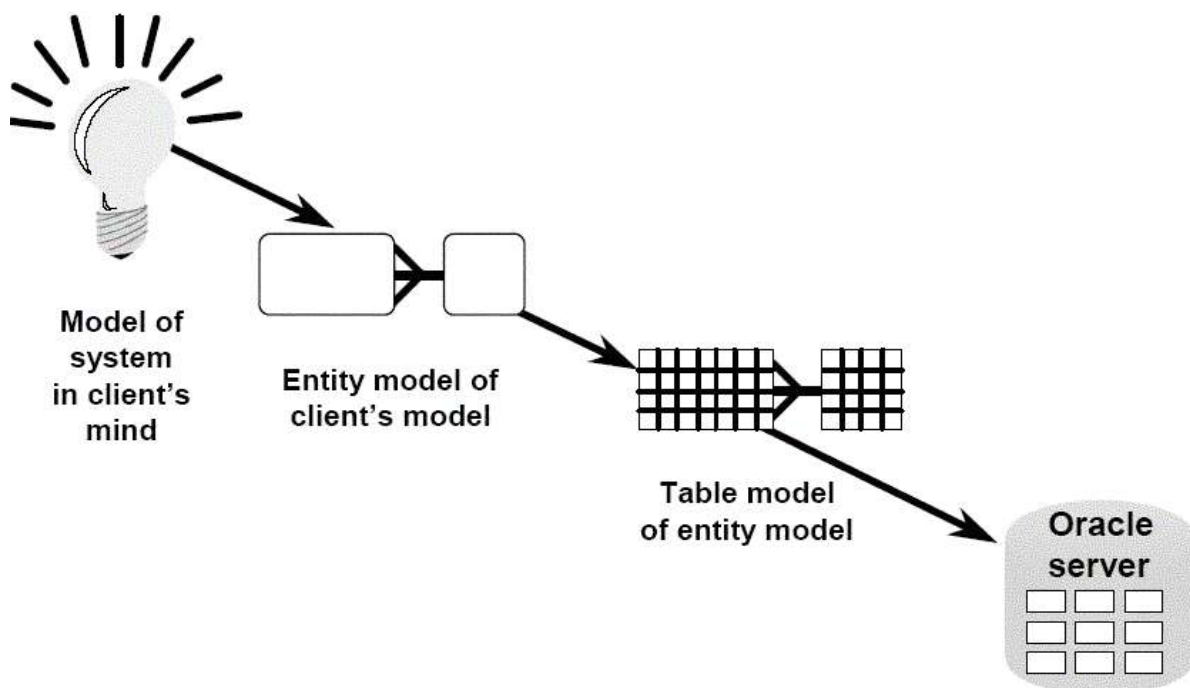
1.4. Relational Database Concept

1.4.1. Definition of a Relational Database



A relational database uses relations or two-dimensional tables to store information. For example, you create different tables to store information about employees, such as an employee table, a department table, and a salary table.

1.4.2. Data Models



Models are used to explore ideas and improve the understanding of the database design.

The main objective of models is to produce a model that fits a multitude of these uses, can be understood by every user, and contains detail for a developer to build a database system.

1.4.3.Entity Relationship Model

Entity-Relationship model separates the information required by a business from the activities performed within a business. Although business can change nature of their activities, the type of information tends to remain constant. Therefore, the data structures also tend to be constant.

1.4.4.Entity Relationship modelling conventions

Entities:

- Soft box with any dimensions
- Singular, unique entity name
- Entity name in uppercase
- Optional synonym names in uppercase within parentheses: ()

Attributes:

- Use singular names in lowercase
- Tag mandatory attributes , or values that must be know , with an asterisk : *
- Tag optional attributes , or values that may be known ; with the letter o

Relationships:

- A label, taught by or assigned by
- An optionally either must be or may be
- A degree , either one and only or on or more

Unique identifiers:

- Unique identifiers can be a combination of attributes or relationship, or both.
- Tag each attributes that is part o the UID with a number symbol : #
- Tag secondary UIDs with a number sign in parentheses :(#)

1.4.5.Relating multiple tables

Table Name: **EMPLOYEES**

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
174	Ellen	Abel	80
142	Curtis	Davies	90
102	Lex	De Haan	90
104	Bruce	Ernst	60
202	Pat	Fay	20
206	William	Gietz	110



Primary key



Foreign key

Table Name: **DEPARTMENTS**

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1900
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700



Primary key

Multiple tables contain data that describes exactly one entity.

Data about different entities can be stored in different tables so, you may need to combine two tables or more to retrieve particular information.

For example, you may want to know the name of a department where a specific employee works. Thanks to an RDBMS you can have this type of information by joining tables using foreign keys.

A foreign key is a column or a set of columns that refer to a primary key in the same table or in another table.

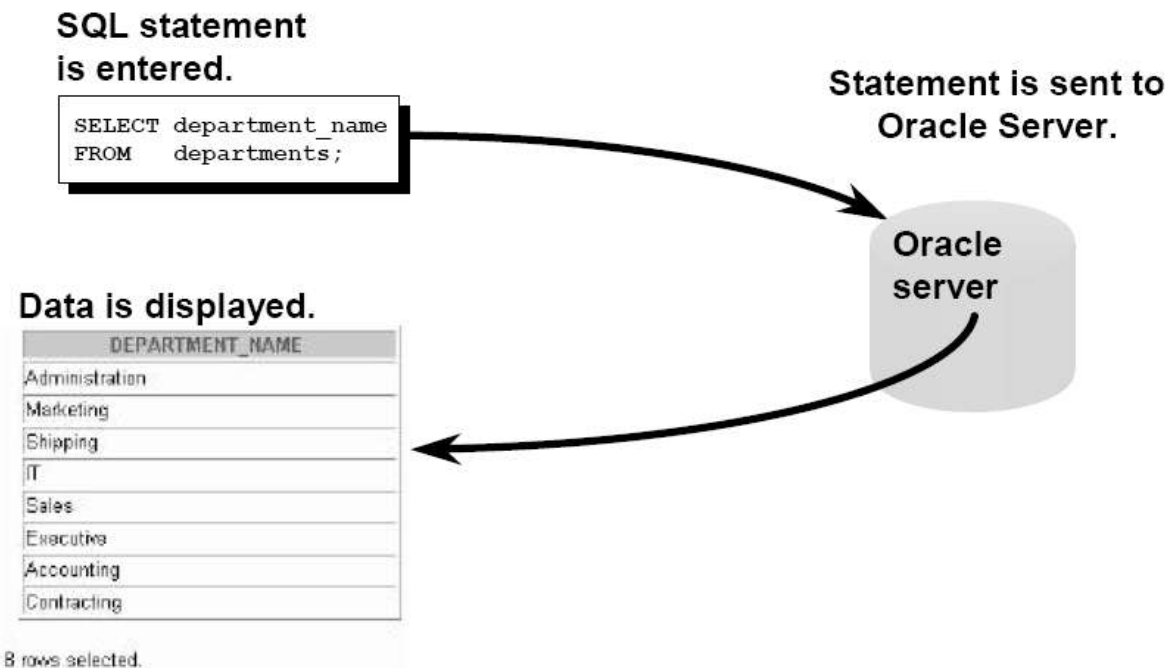
Guidelines for primary keys and foreign keys:

- The value of a primary key is unique
- Primary keys generally cannot be changed
- A foreign key value must be the same than an existing primary key value
- A foreign key must reference either a primary key or unique key column

Properties of a Relational Database:

- You do not specify the access route to the tables
- You do not need to know how the data is arranged physically
- The access to the database is possible , thanks to structured query language (SQL), which is the American National Standards Institute (ANSI)

1.5. Communicating with a RDBMS using SQL



You can communicate with the Oracle Server using the SQL. This language is efficient, easy to learn and to use.

Oracle9i is a flexible RDBMS. Using it, you can store and manage data with all the advantages of a relational structure.

Oracle9i supports Java and XML.

An Oracle Server consists of an Oracle database and an Oracle server instance. Every time a database is started, a system global area (SGA) is allocated, and Oracle background processes are started. The combination of the background processes and memory buffers is called an Oracle instance.

1.6. SQL Statements

Statement	Type	Description
<i>SELECT</i>	DRL Data Retrieval Language	Retrieves data from the database.
<i>INSERT</i> <i>UPDATE</i> <i>DELETE</i> <i>MERGE</i>	DML Data Manipulation Language	Enters new rows, changes existing rows, and removes unwanted rows from tables in the database, respectively.
<i>CREATE</i> <i>ALTER</i> <i>DROP</i> <i>RENAME</i> <i>TRUNCATE</i>	DDL Data Definition Language	Sets up, changes, and removes data structures from tables.
<i>COMMIT</i> <i>ROLLBACK</i> <i>SAVEPOINT</i>	TCS Transaction Control Statement	Manages the changes made by DML statements. Changes to the data can be grouped together into logical transactions.
<i>GRANT</i> <i>REVOKE</i>	DCL Data Control Language	Gives or removes access rights to both Oracle database and the structures within are.

2. Writing Basic SQL SELECT statements

2.1. SELECT statement

2.1.1. Capabilities of SQL SELECT statement

Projection

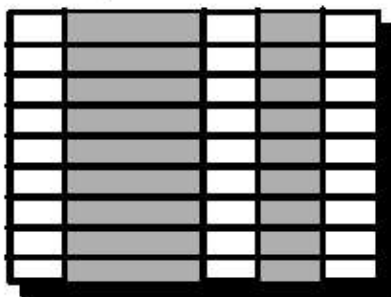


Table 1

Selection

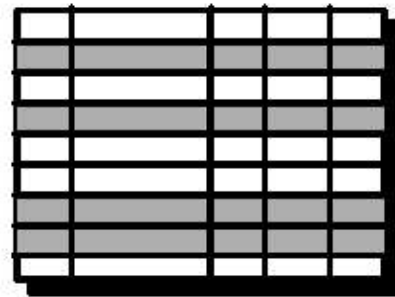


Table 1

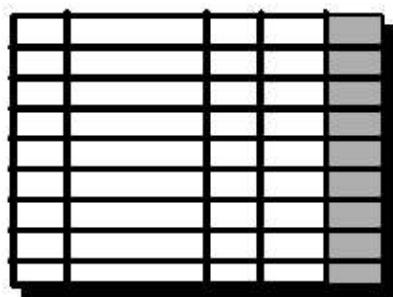


Table 1

Join

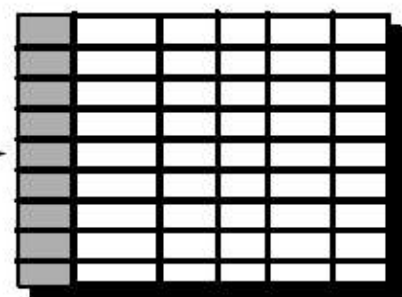


Table 2

A *SELECT* statement is used to retrieve information from the database.

Using the *SELECT* statement, you can do the following:

- Projection : Choose columns that you want returned by the query.
- Selection : Choose the rows in a table you want returned by the query.
- Joining : Bring together data that is stored in different tables by creating a link between them.

2.1.2. Basic SELECT statement

Here are the different parts of a SELECT statement:

SELECT	Is a list of one or more columns
*	Select all columns from the table
DISTINCT	Suppresses duplicates
<i>Column</i> <i>expression</i>	Selects the named column or the expression
<i>[Alias]</i>	Gives selected columns different headings
FROM <i>table</i>	specifies the table containing the columns

The column's order specifies in the **SELECT** list, is the order in which the columns will be displayed.

When displayed, numerical values are automatically placed at the right, and others at the left.

2.1.3. Selecting columns

Thanks to the **SELECT** statement you can choose to display all columns of data in a table or only specific columns.

Selecting all columns of all rows:

Using the **SELECT** keyword with an asterisk, you can display all columns of data in a table, but you can do it in a different way by listing all the columns after the **SELECT** keyword.

Here are two different examples that allow displaying all columns in the departments table:

```
SQL> SELECT *
2 FROM dept;

DEPTNO DNAME LOC
-----
10 ACCOUNTING NEW YORK
20 RESEARCH DALLAS
30 SALES CHICAGO
40 OPERATIONS BOSTON
```

```
SQL> SELECT deptno, dname, loc
2 FROM dept;

DEPTNO DNAME LOC
-----
10 ACCOUNTING NEW YORK
20 RESEARCH DALLAS
30 SALES CHICAGO
40 OPERATIONS BOSTON
```

Selecting specific columns:

The **SELECT** statement can be used to display specific columns of a table by specifying the column names, separated by commas in the statement.

In the **SELECT** clause, we have to specify the column names we want, in the order in which we want them to appear in the screen.

Here is an example of a specific selection:

```
SQL> SELECT  dname,deptno
      2  FROM dept;

DNAME          DEPTNO
-----
ACCOUNTING      10
RESEARCH        20
SALES           30
OPERATIONS      40
```

2.1.4. Writing SQL statements

To have valid and easy to manage SQL statements, you have to use the following rules:

- SQL statements are not case sensitive
- SQL statements can entered on one or many lines
- Keyword can not be split across lines or abbreviated
- Clauses are usually placed on separate lines
- Indents should be used to make code easy to read
- Keywords are typically entered in uppercase

On iSQL*PLUS, you just have to click on the Execute button to run commands or SQL statements.

2.1.5. Arithmetic expressions

Sometimes, you may need to perform calculations or, modify the way in which data is displayed. An arithmetic expression can contain column names, constant numeric values, and the arithmetic operators.

OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide

Using Arithmetic operators:

In this SQL statement, the addition operator is used to increase the salary of all employees of 300\$, and displays a new SAL + 300 column.

The resultant calculated column, is not a new column in the EMP table, but just for display.

```
SQL> SELECT ename, sal+300
2  FROM emp;

      ENAME      SAL+300
-----
SMITH          1100
ALLEN          1900
WARD           1550
JONES          3275
MARTIN         1550
BLAKE          3150
CLARK          2750
...
17 ligne(s) sélectionnée(s).
```

Operator precedence:

If the SQL statement contains more than one arithmetic operator, multiplication and division are evaluated first, but if the operators are of same priority, evaluation is done from left to right.

You can also use parentheses to force the expression within parentheses to be evaluated first.

In this example, the statement displays the name, salary and the annual salary of employees plus a bonus of 100\$.

The multiplication is performed first.

```
SQL> SELECT ename,sal,sal*12+100
2  FROM emp;

      ENAME      SAL  SAL*12+100
-----
SMITH          800      9700
ALLEN         1600     19300
WARD          1250     15100
JONES         2975     35800
MARTIN        1250     15100
BLAKE         2850     34300
...
17 row(s) selected.
```

Using parentheses:

You can use parentheses to specify the order in which the operators are executed.

In this example, the SQL statement displays the name, the salary, and an annual compensation of employees, which is calculated with a monthly bonus of 100\$.

```
SQL> SELECT ename, sal, 12*(sal+100)
2  FROM emp;

  ENAME          SAL  12*(SAL+100)
-----
SMITH             800      10800
ALLEN            1600     20400
WARD             1250     16200
JONES            2975     36900
MARTIN           1250     16200
BLAKE            2850     35400
...
17 row(s) selected.
```

Null values in arithmetic expressions:

Columns of any data can contain null values. A null value is not a zero or a space.

If any null value is returned by an arithmetic expression, all the result is null.

If you attempt to divide with zero an error occurs, but if you divide a number by a null value the result is null or unknown.

In the example, some of employees' don't get commission, so the result of the arithmetic expression is null.

```
SQL> SELECT ename, sal*comm
2  FROM emp;

  ENAME          SAL*COMM
-----
SMITH            2400000
ALLEN             800000
WARD             625000
JONES
MARTIN           1750000
BLAKE
...
17 row(s) selected.
```

2.2. Customized statements

2.2.1. Using column Aliases

You can specify in the SQL statements, alias that will be displayed instead of the column names. By default alias appear in uppercase and don't contain spaces, but you can enclose them in double quotation marks to choose the case and add special characters.

Syntax:

```
SELECT      column1 AS alias1, column2 AS alias2...
FROM      tablename;
```

Alias: Is the name of the alias given to the column.
Tablename: Is the name of the table.

In this example, the SQL statement displays name and salary of all employees, but instead of the default column names, the column names displayed is replaced by the alias specified in the statement.

Notice that the result of the query is the same whether the **AS** keyword is used or not.

```
SQL> SELECT ename AS name,sal AS salary
2  FROM emp;

NAME            SALARY
-----
SMITH            800
ALLEN            1600
WARD             1250
JONES            2975
MARTIN           1250
BLAKE            2850
...
17 row(s) selected.

SQL> SELECT ename "Name",sal "Salary of employee"
2  FROM emp;

Name            Salary of employee
-----
SMITH            800
ALLEN            1600
WARD             1250
JONES            2975
MARTIN           1250
BLAKE            2850
...
17 row(s) selected.
```

2.2.2.Using concatenation operator

Thanks to concatenation operator, it's possible to link columns to other columns, arithmetic expressions, or constant values to create expression.

In this example, name and job are concatenated, and given the alias Employees.

```
SQL> SELECT ename||job AS employees
2  FROM emp;

EMPLOYEES
-----
SMITHCLERK
ALLENSALESMAN
WARDSALESMAN
JONESMANAGER
MARTINSALESMAN
BLAKEMANAGER
CLARKMANAGER
SCOTTANALYST
KINGPRESIDENT
...
17 row(s) selected.
```


2.2.3.Using literal character string

A literal is a character, a number or a date included in the **SELECT** list and that is not a column name or a column alias.

Literal strings of free-format text can be included in the query result and are treated as a column in the **SELECT** list.

Date and character literals **MUST** be enclosed within single quotation marks.

In this example, the SQL statement displays a sentence which is the result of a concatenation of the name, and the salary columns.

```
SQL> SELECT ename||': 1 Month salary = '||sal AS Monthly
2 FROM emp;

MONTHLY
-----
SMITH: 1 Month salary = 800
ALLEN: 1 Month salary = 1600
...
17 row(s) selected.
```

2.2.4.Eliminating duplicate rows

If don't indicate it, iSQL*PLUS displays the results of a query without eliminating duplicate rows. But it's possible, thanks to the **DISTINCT** keyword used immediately after the **SELECT** keyword, to eliminate all duplicated rows.

Syntax:

```
SELECT      [DISTINCT] { * | column [alias] | expr ... }
FROM        tablename;
```

Multiple columns can be specified after the **DISTINCT** keyword.

In this example, the SQL statement displays all distinct department names.

```
SQL> SELECT DISTINCT dname
2 FROM dept;

DNAME
-----
ACCOUNTING
OPERATIONS
RESEARCH
SALES
STORE
5 row(s) selected.
```

2.3. SQL and iSQL*PLUS interaction

2.3.1.SQL statements AND iSQL*PLUS commands

SQL is a command language for communication with the Oracle Server.

iSQL*PLUS is an Oracle tool that recognizes and submits SQL statements to the Oracle Server for execution and contains its own command language.

Features of SQL:

- Can be used by a range of users.
- Nonprocedural language.
- Reduces the amount of time required for creating and maintaining systems
- English-like language.

Features of iSQL*PLUS:

- Accessed from a browser.
- Accepts ad hoc entry of statements.
- Provides online editing for modifying SQL statements.
- Controls environmental settings.
- Format query results into a basic report.
- Accesses local and remote databases.

The following table compares SQL and iSQL*PLUS:

SQL	iSQL*PLUS
Is a language for communicating with the Oracle Server to access data	Recognizes SQL statements and sends them to the server
Is based on ANSI standard SQL	Is the Oracle proprietary interface for executing SQL statements
Manipulates data and table definitions in the database	Does not allow manipulation of values in the database
Does not have a continuation character	Has a dash (-) as a continuation character if the command is longer than one line
Cannot be abbreviated	Can be abbreviated
Uses functions to perform some formatting	Uses commands to format data

2.3.2.Using iSQL*PLUS

iSQL*PLUS is an environment in which we can execute SQL statements, format and perform calculations, create scripts files to store SQL statements.

iSQL*PLUS commands can be divided into the following categories:

Category	Purpose
Environment	Affects the general behaviour of SQL statements for the session

Format	Formats query results
File manipulation	Saves statements into text script files , and runs statements from text scripts files
Execution	Sends SQL statements from the browser to Oracle Server
Edit	Modifies SQL statements in the Edit Window
Interaction	Allows you to create and pass variables to SQL statements, print variables values, and print messages to the screen
Miscellaneous	Has various commands to connect to the database, manipulate the iSQL*PLUS environment, and display column definitions

2.3.3.Displaying table structure using DESCRIBE command

It's possible to display the structure of a table using the **DESCRIBE** command.

This command shows the column names and data types, as well as whether a column **MUST** contain data.

Name: Indicates the column names.

Null?: Indicates if the column can contain null values

Type: Displays the data types

In this example, the **DESCRIBE** command displays the structure of the EMP table.

SQL> DESCRIBE emp;		
Name	NULL ?	Type
-----		-----
EMPNO	NOT NULL	NUMBER (4)
ENAME		VARCHAR2 (10)
JOB		VARCHAR2 (9)
MGR		NUMBER (4)
HIREDATE		DATE
SAL		NUMBER (7, 2)
COMM		NUMBER (7, 2)
DEPTNO		NUMBER (2)

The data types are described in the following table:


Data Type	Description
NUMBER(p,s)	Number value having a maximum number of digits p, with s digits to the right of the decimal point.
VARCHAR2(s)	Variable-length character value of maximum size s
DATE	Date and time value between January 1, 4712 B.C., and December 31,9999 A.D..
CHAR(s)	Fixed-length character value of size s

2.3.4.Interacting with script files on iSQL*PLUS

Logging in to iSQL*PLUS:



To log in through a browser environment, you just have to enter the login, the password and the connection identifier, then to click on the connection button.



Using statements and commands from a script file in iSQL*PLUS:

- 1) Use this button to find the script name and location you want to use.
- 2) The file contents are loaded into the iSQL*PLUS edit window.
- 3) Save the text present into the edit window, in a script file.
- 4) Gives you a list of the recent used scripts

2.3.5.Interacting with script files using commands

The SQL*PLUS commands are used to save, load and execute script files.

Commands	Description	Example
SAV[E] filename[.ext] [RE[PLACE]] APP[END]]	Save the buffer contents into a file. APPEND specifies that the buffer contents must be added to an existing file. REPLACE specifies that the existing file must be overwritten.	<pre>SQL>SAV all_emp</pre>
GET filename[.ext]	Allows getting the buffer contents which has been saved into a file.	<pre>SQL> GET all_emp</pre>
STA[RT] filename[.ext]	Allows executing the file contents	<pre>SQL> STA all_emp</pre>
@ filename[.ext]	Allows execute the file contents	<pre>SQL> @all_emp</pre>
ED[IT]	Allows calling the editing tool, and save the buffer contents into the afiedt.buf files.	<pre>SQL> ED all_emp</pre>
ED[IT] [filename[.ext]]	Allows editing a file contents thanks to an editing tool.	<pre>SQL> ED all_emp</pre>
SPO[OL] filename[.ext] [OFF OUT]	Allows saving statement results into a file. OFF close the spool files. OUT close the spool files and send the results to printer.	<pre>SQL> SPO all_emp</pre>
EXIT	Allows disconnecting and closing SQL*PLUS automatically	<pre>SQL> EXIT</pre>

2.3.6.SQL*PLUS editing commands

The SQL*PLUS editing commands allow manipulate the buffer contents.

Command	Description	Example
A[PPEND] text	Allows adding in the last line of the buffer the string <i>text</i>	<pre>SQL>SELECT ename,deptno</pre> <p>The buffer contains: SELECT ename,deptno.</p> <pre>SQL> A job</pre> <p>The buffer contains: SELECT ename, deptno, job.</p>
C[HANGE] /old /new	Allows replacing the string old from the buffer by the string new	<pre>SQL> C /job/sal</pre> <p>The buffer contains SELECT ename,deptno,sal</p>
C[HANGE] / old /	Allows deleting the string old from the buffer.	<pre>SQL> C/,sal</pre> <p>The buffer contains SELECT ename,deptno</p>
I[INPUT]	Allows inserting a new line in the buffer	<pre>SQL>SELECT ename,deptno,sal</pre> <p>The buffer contains SELECT ename,deptno,sal</p>
I[INPUT]	Allows inserting a new line in which contains the string text in the buffer	<pre>SQL> I FROM emp</pre> <p>The buffer contains SELECT ename,deptno,sal FROM emp</p>
L[IST]	Allows displaying the buffer contents	<pre>SQL> L</pre>
L[IST] n	Allows displaying the line n from the buffer	
L[IST] m n	Allows displaying the line m to n from the buffer	
R[UN]	Allows executing the buffer contents. This command display the statement followed by its result	<pre>SQL> RUN</pre>
n	Allows specifying the current line	<pre>SQL> 1 1 SELECT ename,deptno,sal SQL> 1 SELECT mgr</pre> <p>The buffer contains SELECT mgr FROM emp</p>
n text	Allows replacing the line n by text	
0 text	Allows inserting a line containing text before the line number 1	
DEL	Erases the current line from the buffer	<pre>SQL> DEL 1 2</pre>
DEL n	Erases the line n from the buffer	
DEL m n	Erases lines m to n from the buffer	
CL[EAR] BUFF[ER]	Erases the buffer contents	<pre>SQL> CLEAR BUFFER</pre>

3.Restricting and storing data

3.1. Limiting rows using a selection

3.1.1.Using the WHERE clause

You can restrict the rows returned from the query by using the **WHERE** clause, which contains a condition that must be met.

Syntax:

```
SELECT    column [alias]
FROM      tablename
WHERE     condition.
```

WHERE Restricts the query to rows meet a condition

CONDITION Is composed of column names, expressions, constants, and a comparison operator.

The **WHERE** clause can compares values in columns, literal values, arithmetic expressions, or functions.

In this example, the SQL statement retrieves the name, job, and department id of the employee number 7902:

```
SQL> SELECT ename, job, deptno
2 FROM emp
3 WHERE empno=7902;
```

ENAME	JOB	DEPTNO
FORD	ANALYST	20

The following statement displays the name, job, and department number of all employees working as ANALYST.

```
SQL> SELECT ename, job, deptno
2 FROM emp
3 WHERE job='ANALYST';
```

ENAME	JOB	DEPTNO
SCOTT	ANALYST	20
FORD	ANALYST	20

This statement displays the name, the job, and the department number of all employees hired before the 23-JAN-82.

```
SQL> SELECT ename, job, deptno
2 FROM emp
3 WHERE hiredate = '23-JAN-82';
```

ENAME	JOB	DEPTNO
MILLER	CLERK	10

3.1.2.Using WHERE clause with different data types

Character strings and dates:

You just have to know that character strings and dates must be enclosed in single quotation marks(' '), and that all character strings are case sensitive, and the default date display is DD-MON-RR.

In this example, the SQL statement displays the job of the employee called KING.

```
SQL> SELECT job
2 FROM emp
3 WHERE ename='KING';

JOB
-----
PRESIDENT
```

3.2.Comparison operators

3.2.1.Arithmetic operator

Comparison conditions are used in conditions that compare one expression to another value expression.

They are used in the **WHERE** clause.

Syntax:

... *WHERE* expr operator value

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

In this example, the SQL statement displays the name of employees that earn more than 4000\$:

```
SQL> SELECT job
2 FROM emp
3 WHERE SAL>4000;

ENAME
-----
KING
```

3.2.2.Using the BETWEEN condition

You can display rows based on a range of values using the **BETWEEN** range condition. The range that you specify contains a lower limit and an upper limit.

Syntax:

WHERE *column_name* **BETWEEN** *lower_limit* **AND** *higher_limit*

In this example, the SQL statement displays name of the employees that earn more than 1000\$ and less than 3000\$:

```
SQL> SELECT ename
2  FROM emp
3  WHERE sal BETWEEN 1000 AND 3000;

ENAME
-----
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT
...
12 row(s) selected.
```

You can also use the **BETWEEN** condition with dates.

```
SQL> SELECT      ename, hiredate
2  FROM          emp
3  WHERE          hiredate BETWEEN '01-JAN-81' AND '31-JUL-
81';

ENAME      HIREDATE
-----
ALLEN      20/02/81
WARD       22/02/81
JONES      02/04/81
BLAKE      01/05/81
CLARK      09/06/81
```

3.2.3.Using the LIKE condition

If you don't know the exact value to search, you can use the **LIKE** condition to select rows that match a character pattern.

You can use two symbols for the search:

Symbol	Description
%	Represents any sequences of zero or more characters
-	Represents any single character

Syntax:

WHERE *column_name* **LIKE** 'value including wildcards'

This statement displays the job of all employee that have a name beginning with S.

```
SQL> SELECT ename, job
2  FROM emp
3  WHERE ename LIKE 'S%';
```

ENAME	JOB
SMITH	CLERK
SCOTT	ANALYST

You can combine the `_` and `%` symbols. This example displays the job and names of all employees whose names have a `L` as the second character.

```
SQL> SELECT ename, job
2 FROM emp
3 WHERE ename LIKE '_L%';
```

ENAME	JOB
ALLEN	SALESMAN
BLAKE	MANAGER
CLARK	MANAGER

If you need to have an exact match for the actual `%` and `_` characters, you can use the **ESCAPE** option, which specifies what the escape character is.

Syntax:

WHERE *column_name* **LIKE** 'value1value2' **ESCAPE**
'escapecharacter'

In this example, we want to display names beginning with the character strings "SGBD_". Thanks to the **ESCAPE** option, the backslash (`\`) is identified as the escape character, so the underscore (`_`) is interpreted as a literal character.

```
SQL> SELECT ename
2 FROM emp
3 WHERE ename LIKE 'SGBD\_%' ESCAPE '\\';
```

ENAME
SGBD_JAROD
SGBD_HELYOS
SGBD_YO

3.2.4.Using the IN condition

The **IN** condition allows you to test values in a specified set of other values.

Syntax:

WHERE *column_name* **IN** (value1, value2, value3...)

The example displays employee's number, name and manager's employee numbers for all employees whose manager's employee numbers is 7782, 7698, or 7839.

```
SQL> SELECT empno, ename, mgr
2 FROM emp
3 WHERE mgr IN (7782, 7698, 7839);
```

EMPNO	ENAME	MGR
7499	ALLEN	7698

```

7521 WARD          7698
7566 JONES         7839
7654 MARTIN        7698
7698 BLAKE         7839
7782 CLARK         7839
7844 TURNER        7698
7900 JAMES         7698
7934 MILLER        7782
7952 PAPIER        7839
10 row(s) selected.

```

The **IN** condition can be used with characters or dates; you just have to enclose it in simple quotation marks. (' ').

```

SQL> SELECT job,ename
2   FROM emp
3   WHERE ename IN ('ALLEN','WARD');

```

JOB	ENAME
SALESMAN	ALLEN
SALESMAN	WARD

3.2.5.Using the NULL condition

This condition includes the **IS NULL** condition and the **IS NOT NULL** condition. The **IS NULL** condition tests for nulls and the **IS NOT NULL** condition test for the not nulls. Notice that is impossible to use = with null values.

Syntax:

WHERE *column_name* IS NULL

This statement displays all employees with a commission.

```

SQL> SELECT ename,sal,comm
2   FROM emp
3   WHERE comm IS NOT NULL;

```

ENAME	SAL	COMM
SMITH	800	3000
ALLEN	1600	500
WARD	1250	500
MARTIN	1250	1400
TURNER	1500	0
ADAMS	1100	3000
JAMES	950	3000
FORD	3000	300
MILLER	1300	3000
PAPIER	1000	0

10 row(s) selected.

3.3.Logical conditions

3.3.1.Using the AND operator

With the **AND** operator, both conditions must be true for any records to be selected.

AND	TRUE	FALSE	NULL
-----	------	-------	------

TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Syntax:

WHERE *condition1*
AND *condition2*;

This statement displays names of all employees that don't have a commission and don't have any manager.

```
SQL> SELECT ename
2 FROM emp
3 WHERE comm IS NULL
4 AND mgr IS NULL;

ENAME
-----
KING
THG
```

3.3.2.Using the OR operator

With the **OR** operator, if one or more conditions is true the row is selected.

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Syntax:

WHERE *condition1*
OR *conditon2* ;

This statement , displays names of all employees that don't have commission or have a salary higher than 3000\$.

```
SQL> SELECT ename
2 FROM emp
3 WHERE comm IS NULL
4 OR sal>3000;

ENAME
-----
JONES
BLAKE
CLARK
SCOTT
KING
THG
```

3.3.3.Using the NOT operator

This condition is false the row is selected.

NOT	TRUE	FALSE	NULL
-----	------	-------	------

	FALSE	TRUE	NULL
--	-------	------	------

Syntax:

WHERE *column_name* **NOT** *comparison_operator* *value*

Example:

```
... WHERE empno NOT IN ...  
... WHERE sal NOT BETWEEN ...  
... WHERE ename NOT LIKE ...  
... WHERE mgr IS NOT NULL...
```

This statement displays employee names whose don't earn 1000\$ or 5000\$ or 2000\$.

```
SQL> SELECT ename  
2 FROM emp  
3 WHERE sal NOT IN (5000,1000,2000);  
  
ENAME  
-----  
SMITH  
ALLEN  
WARD  
JONES  
MARTIN  
BLAKE
```

3.3.4.Rules of precedence

Order Evaluated	Operator
1	Arithmetic operators
2	Concatenation operator
3	Comparison contions
4	IS [NOT] NULL,LIKE,[NOT] IN
5	[NOT] BETWEEN
6	NOT logical condition
7	AND logical condition
8	OR logical condition

SQL>	SELECT	ename, job, sal
2	FROM	emp
3	WHERE	(job='SALESMAN'
4	OR	job='PRESIDENT')
5	AND	sal>1500;
ENAME	JOB	SAL
-----	-----	-----
ALLEN	SALESMAN	1600
KING	PRESIDENT	5000

3.4.Sorting results

3.4.1.Using the ORDER BY clause

The **ORDER BY** clause allows you to sort rows. This clause must be the last of the SQL statement. It's possible to specify expression, alias or column positions as the sort condition.

Syntax:

```
SELECT      expr
FROM        table
[WHERE      condition(s)]
[ORDER BY   {column,expr} [ASC,DESC]];
```

ORDER BY: Specifies the order in which the retrieved rows are displayed
ASC: Orders the rows in ascending order
DESC: Orders the rows in descending order.

When the **ORDER BY** clause is not used, the sort order is undefined.

3.4.2.Sorting in descending order

The default sort order is ascending, but it's possible to inverse the order by using the **DESC** keyword. Sort order can be used on dates, numbers and characters values. Notice that null values are always displayed first for descending sequences.

This statement displays employee names, and their salary, ordering the result by salary in a descending order:

SQL>	SELECT	ename, sal
2	FROM	emp
3	ORDER BY	sal DESC
ENAME	SAL	
-----	-----	
THG		
KING	5000	
SCOTT	3000	
FORD	3000	
JONES	2975	
BLAKE	2850	

3.4.3.Sorting by column alias

You can use a column alias in the **ORDER BY** clause.

Here is an example:

```
SQL> SELECT ename,sal a
2  FROM emp
3  ORDER BY salary;

  ENAME          SALARY
-----
SMITH              800
JAMES              950
PAPIER            1000
ADAMS             1100
WARD              1250
MARTIN            1250
MILLER            1300
TURNER            1500
```

3.4.4.Sorting by multiple columns

You can use the **ORDER BY** clause, to sort by multiple columns.

You just have to specify the columns and separate its name using commas.

Notice that it's also possible to include columns that are not into the **SELECT** clause.

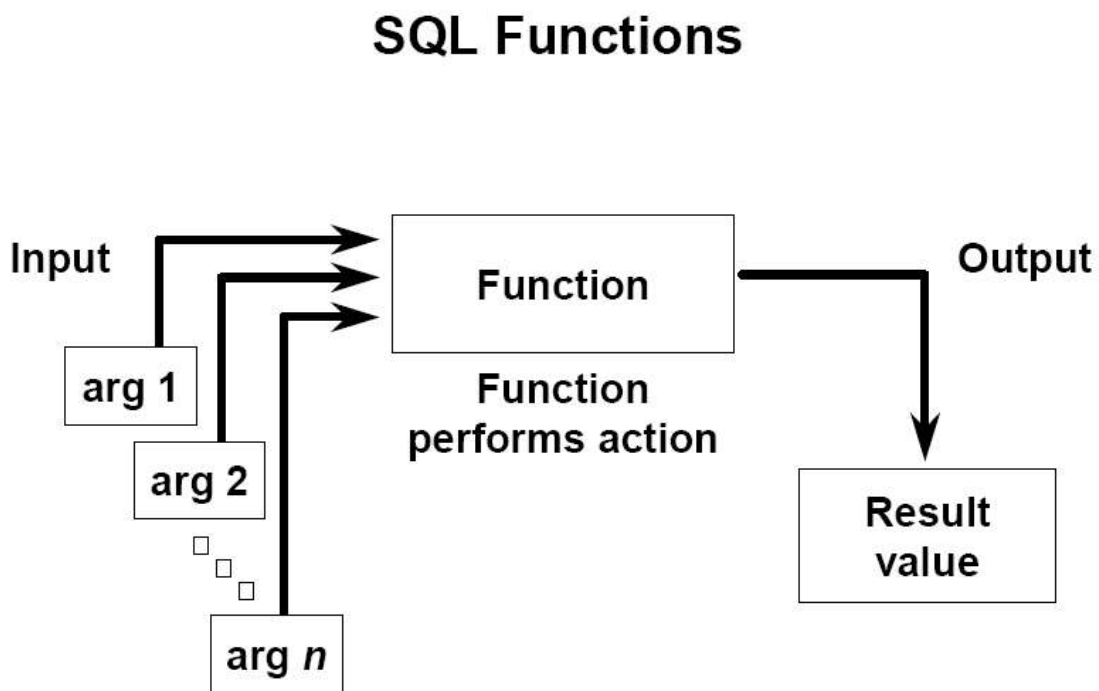
This statement displays employee salaries , and employee names, ordered by department id , and then in descending by salary.

```
SQL> SELECT ename,sal
2  FROM emp
3  ORDER BY deptno,sal desc;

  ENAME          SAL
-----
KING              5000
CLARK             2450
MILLER            1300
SCOTT             3000
FORD              3000
JONES             2975
ADAMS             1100
```

4. Single-row functions

4.1. SQL functions



Functions are very powerful feature of SQL and can be used to:

- Perform calculations on data.
- Modify individual data items.
- Manipulate output for groups of rows.
- Format dates and numbers for display.
- Convert column data types.

These functions operate on single row only and return one result per row.
There are four main types of functions:

- a. Character.
- b. Number.
- c. Date.
- d. Conversion.

4.2. Character functions

4.2.1. Case manipulation functions

Oracle gives some Case manipulation functions that allows converting character cases.

Function	Description	Example	Result
LOWER	Converts mixed case or uppercase character strings to lower case	LOWER('ORACLE CourSe')	Oracle course
UPPER	Converts mixed case or lowercase character strings to uppercase	UPPER('oRaCle Course')	ORACLE COURSE
INITCAP	Converts the first letter of each word to uppercase and remaining letters to lowercase	INICATP('I love ORACLE')	I love oracle

Notice that text **MUST** be enclosed in simple quotation marks and parentheses. (' ').

Here is an example:

```
SQL> SELECT 'The employee ' || INITCAP(ename) || ' earns ' || sal
2  example
3  FROM emp
4  WHERE ename='KING'

EXAMPLE
-----
The employee King EARNs 5000
```

4.2.2. Character-manipulation functions

Single-Row character functions accept character data as input and can return number or character.

Function	Description	Example	Result
CONCAT	Joins values together	CONCAT('Hello','world')	Hello world
SUBSTR	Extracts a string of determined length	SUBSTR('HelloWorld',1,5)	Hello
LENGTH	Shows the length of a string as a numeric value	LENGTH('ORACLE')	6
INSTR	Finds numeric position of a named character	INSTR('Helloworld','w')	6

LPAD	Pads the character value right-justified	LPAD(salary,10,'*')	*****24000
RPAD	Pads the character value left-justified	RPAD(salary,10,'*')	24000*****
TRIM	Trims heading or trailing characters from a character string	TRIM('H' FROM 'Hello')	ello

Here is an example:

```
SQL> SELECT ename, CONCAT (ename, job), LENGTH(ename),
2 INSTR(ename, 'A')
3 FROM emp
4 WHERE SUBSTR(job,1,5) = 'SALES';

ENAME CONCAT(ENAME,JOB) LENGTH(ENAME) INSTR(ENAME,'A')
-----
MARTIN MARTINSALESMAN 6 2
ALLEN ALLENSALESMAN 5 1
TURNER TURNERSALESMAN 6 0
WARD WARDSALESMAN 4 2
```

4.3. Number functions

4.3.1. Using the ROUND function

The **ROUND** function rounds the column, expression or value to n decimal places. If the second argument is equal to 0 or is missing the value is rounded to zero decimal places.

Syntax:

ROUND (column expr [,n])	Allows rounding a value <i>column</i> or value of <i>expr</i> to <i>n</i> digits.
-----------------------------------	---

If the second argument is 2, the value is rounded to two decimal places, and if the second argument is -2, the value is rounded to two decimal places to the left.

Notice that this function can be used with dates.

```
SQL> SELECT ROUND(45.923,2), ROUND(45.923,0), ROUND(45.923,62)
2 FROM dual;

ROUND(45.923,2) ROUND(45.923,0) ROUND(45.923,-1)
-----
45,92 46 50
```

4.3.2. Using the TRUNC function

The **TRUNC** function truncates the column, expression, or value to n decimal places. This function works like the **ROUND** function, except that the value is truncated instead of rounded.

Syntax:

<http://www.labo-oracle.com>

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

TRUNC(column expr [,n])	Allows truncating a value <i>column</i> or a value of <i>expr</i> to <i>n</i> digits.
----------------------------------	---

```
SQL> SELECT TRUNC (45.923,2), TRUNC (45.923,0), TRUNC (45.923,62)
2 FROM dual;

TRUNC (45.923,2) TRUNC (45.923,0) TRUNC (45.923,-2)
-----
45,92          45          0
```

4.3.3.Using the MOD function

The **MOD** function finds the remainder of value1 divided by value2.

Syntax:

MOD(m,n)	finds the remainder of <i>m</i> divided by <i>n</i>
-----------------	---

The function can be used to determine if a value is odd or even.

```
SQL> SELECT MOD (14,6)
2 FROM dual;

MOD (14,6)
-----
2
```

4.4. Working with dates

4.4.1.The SYSDATE function

SYSDATE is a function that returns the current database Server date and time. It can be used just as you would use any other column.

SYSDATE	Returns the current date.
----------------	---------------------------

For example, if you want to display the current date you just have to execute the following statement:

```
SQL> SELECT sysdate
2 FROM dual;

SYSDATE
-----
04/08/04
```

4.4.2.Using arithmetic operators with dates

You can perform calculations using arithmetic's operators such as addition and subtractions.

Operation	Result	Description
Date + Number	Date	Adds a number of days to a date
Date – Number	Date	Subtracts a number of days to a date
Date – Date	Number of days	Subtracts one date from another
Date + Number/24	Date	Adds number of hours to a date

This statement displays the number of days between the current date and the employee hire date.

```
SQL> SELECT ename, (SYSDATE-hiredate) days
2 FROM emp;
```

ENAME	DAYS
-----	-----
SMITH	8631,71626
ALLEN	8566,71626
WARD	8564,71626
JONES	8525,71626
MARTIN	8346,71626
BLAKE	8496,71626
CLARK	8457,71626
SCOTT	7909,71626
KING	8296,71626
TURNER	8366,71626
ADAMS	7875,71626

4.4.3.Dates functions

Here are the main date functions:

Function	Description	Example	Result
MONTHS_BETWEEN	Number of months between two dates	MONTHS_BETWEEN('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS	Add calendar months to date	ADD_MONTHS('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY	Next day of the date specified	NEXT_DAY('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY	Last day of the month	LAST_DAY('01-FEB-95')	'28-FEB-95'
ROUND	Returns date rounded to the argument specified	ROUND(SYSDATE,'MONTH') With SYSDATE='25-JUL-95' ROUND(SYSDATE,'YEAR')	'01-AUG-95' '01-JAN-96'

TRUNC	Returns date truncated to the argument specified	TRUNC(SYSDATE,'MONTH') With SYSDATE='25-JUL-95' TRUNC(SYSDATE,'YEAR')	'01-JUL-95' '01-JAN-95'
--------------	--	--	--------------------------------

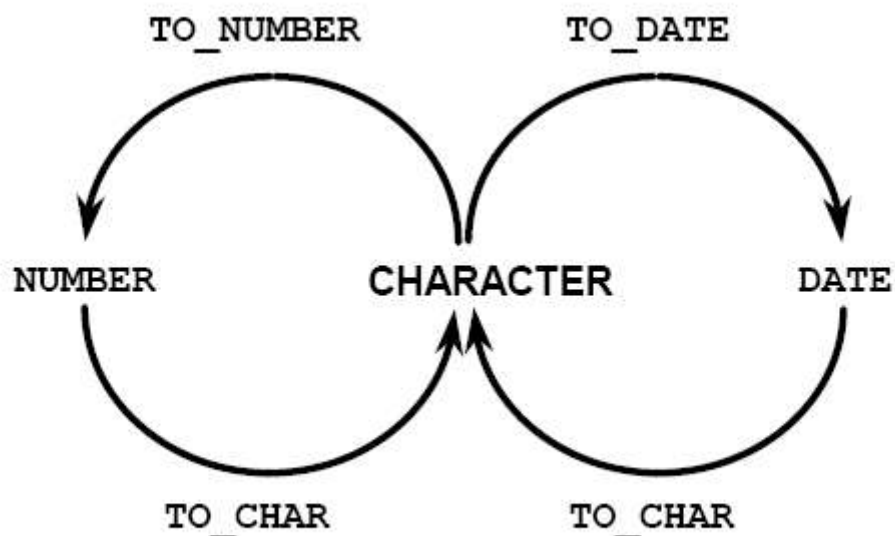
4.5. Conversion functions

4.5.1. Implicit data type conversion

In general, the Oracle server uses the rule for expressions when a data type conversion is needed in places not covered by a rule for assignment conversion.

FROM	TO
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

4.5.2. Explicit data type conversion



SQL provides three functions to convert a value from one data type to another:

Function	Purpose
TO_CHAR (number date,[fmt],[nlsparams])	Converts a number or date value to a VARCHAR2 character string with format model fmt. The nlsparams parameter specifies the language in which month and day names and abbreviations are returned.
TO_NUMBER (char,[fmt],[nlsparams])	Converts a character string containing digits to a number in the format fmt. The nlsparams parameter has the same purpose in this function as in the TO_CHAR function for number conversion.
TO_DATE (char,[fmt],[nlsparams])	Converts a character string representing a date to a date value according to the fmt specified.

4.5.3.Using the TO_CHAR function with dates

The **TO_CHAR** function allows you to convert a date from the *DD-MON-YY* format to one specified format.

The format model must be enclosed in simple quotation marks; it can include any valid date format.

This statement displays the employee hire dates in the *MM/YY* date format.

```
SQL> SELECT ename,TO_CHAR(hiredate,'MM/YY') Month_hired
2 FROM emp;
```

ENAME	MONTH
SMITH	12/80
ALLEN	02/81
WARD	02/81
JONES	04/81
MARTIN	09/81

The SQL statement displays the name and hire dates for all employees. The hire date appears as 17 June 1987.

```
SQL> SELECT ename,TO_CHAR(hiredate,'fmDD Month YYYY') hiredate
2 FROM emp;
```

ENAME	HIREDATE
SMITH	17 Decembee 1980
ALLEN	20 February 1981
WARD	22 February 1981
MARTIN	28 September 1981

4.5.4.Elements of the date format model

Here are the different valid date formats:

Format	Description
YYYY	Full year in numbers

YEAR	Year spelled out
MM	Two-digits value for month
MONTH	Full name of the month
DY	Three-letter abbreviation of the day
DAY	Full name of the day
WW or W	Week of year or month
DDD	Day of the year
DD	Day of the month
D	Day of the week
J	Number of days since 31 December 4713 B.C
Q	Quarter of year
MON	Three-letter abbreviation of the month
RM	Roman numeral month
CC	Century
fmDAY	Erases spaces
AM or PM	Meridian indicator
HH /HH12/HH24	Hour of the day
MI	Minutes (0-59)
SS	seconds (0-59)
SSSS	seconds (0-86399)
TH	Ordinal number(DDTH for 4 TH)
SP	Spelled-out ordinal numbers(DDSP for FOUR)
SPTH or THSP	Spelled-out ordinal numbers(DDSPTH for FOURTH)
BC or AD	B.C./D. indicator
B.C. or A.D.	B.C./A.D. indicator with periods

Example:

```
SQL> SELECT      ename,
2              TO_CHAR(hiredate, 'fmDD Month YYYY') HIREDATE
3 FROM          emp

ENAME      HIREDATE
-----
SMITH      17 Décembre 980
ALLEN      20 Février 981
WARD       22 Février 981
JONES      2 Avril 981
MARTIN     28 Septembre 981
BLAKE      1 Mai 981
...
14 ligne(s) sélectionnée(s).
```

4.5.5.Using the TO_CHAR function with numbers

You can convert a value of NUMBER data type to VARCHAR2 data type.

The Oracle Server displays a string of hash signs (#) in place whole number whose digits exceed the number of digits provided in the format model.

Here are the valid format elements:

Element	Description	Example	Result
9	Numeric position(Number of 9s determine display width)	999999	1234
0	Displays leading zeros	099999	001234
\$	Floating dollar sign	\$999999	\$1234
L	Floating local currency symbol	L999999	FF1234
.	Decimal point in position specified	999999.99	1234.00
,	Comma in position specified	999,999	1,234
MI	Minus signs to right(negatives values)	999999MI	1234-
EE	Scientific notation(Format must specify four Es)	99.999EEEE	1.234E+03
PR	Parenthesize negative numbers	999999PR	
V	Multiply by 10 n times (n=number of 9s after V)	9999V99	123400
B	Displays zero values as blank , not 0	B9999.99	1234.00

This statement displays the employee salaries in a specific format using the **TO_CHAR** function:

```
SQL> SELECT TO_CHAR(sal, '$99,999.00') salary
2 FROM emp;

SALARY
-----
   $800.00
  $1,600.00
  $1,250.00
  $2,975.00
  $1,250.00
```

4.5.6.Using the TO_NUMBER and TO_DATE functions

The **TO_NUMBER** and **TO_DATE** functions allow converting character strings to number format or date format.

These two functions have an **fx** modifier, which specifies the exact matching for the character argument and date format model of a **TO_DATE** function.

The **fx** modifier specifies exact matching for the character argument and date format model of **TO_DATE** function:

- The character argument cannot have extra blanks.
- Numeric data in the character argument must have the same number of digits as the corresponding element in the format model.
- Punctuation and quoted text in the character argument must exactly match the corresponding parts of the format model.

String	Date format	Result
'15-JAN-1998'	'fxDD-MON-YYYY'	No error
'1-JAN-1998'	'fxDD-MON-YYYY'	Error

This statement displays the names and hire dates of all employees who joined on May 24, 1999. Using the **fx** modifier, you must have an exact match.

```
SQL> SELECT ename,hiredate
  2 FROM emp
  3 WHERE hiredate=TO_DATE('MAY 24,1999','fxMonth DD,YYYY');
WHERE hiredate=TO_DATE('MAI 24,1999','fxMonth DD,YYYY')
*
ERROR line 3 :
ORA-01858: a non-numeric character wan found where a numeric
was expected
```

The number of spaces does not match, so an error occurs.

4.5.7.RR date format

The **RR** date format can be used to specify different centuries.

You can use the **RR** date format element instead of the **YY**; the century of the return value varies according to the specified two-digit year and the last two digits of the current year.

Current year	Date given	Format RR	Format YY
1995	22-OCT-95	1995	1995
1995	22-OCT-17	2017	1917
2001	22-OCT-17	2017	2017
2001	22-OCT-95	1995	2095

		If the specified two-digit year is:	
		0-49	50-99
If two digits of the current year are	0-49	The return date is in the current century	The return date is in the century before the current one
	50-99	The return date is in the century after the current one	The return date is in the current century

This SQL statement displays the employees hired prior to 1990:

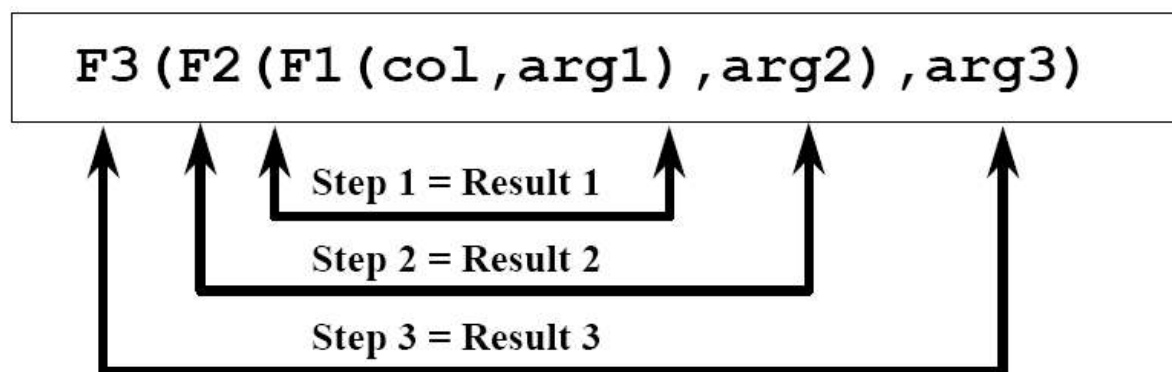
```
SQL> SELECT ename,TO_CHAR(hiredate,'DD-MON-YYYY')
2 FROM emp
3 WHERE hiredate< TO_DATE('01-Jan-90','DD-Mon-RR');
```

ENAME	TO_CHAR(HIR
SMITH	17-DEC-1980
ALLEN	20-FEV-1981
WARD	22-FEV-1981
JONES	02-AVR-1981
MARTIN	28-SEP-1981
BLAKE	01-MAI-1981

4.6.Nesting functions

You can nest single-row functions to any depth you want.

You just have to notice that nested functions are evaluated from the innermost level to the outermost level.



This example displays the head of the company, who has no manager:

1. The **TO_CHAR** function is evaluated
2. The **NVL** function is evaluated to replace null values with a text.

```
SQL> SELECT ename,NVL(TO_CHAR(mgr), 'No manager')
2 FROM emp;

      ENAME      NVL (TO_CHAR (MGR) , 'NOMANAGER')
-----
SMITH          7902
ALLEN          7698
WARD           7698
JONES          7839
MARTIN         7698
BLAKE          7839
CLARK          7839
SCOTT          7566
KING           No manager
TURNER         7698
...
17 row(s) selected.
```

4.7.General functions

4.7.1.Using the NVL function

The **NVL** function is used to convert a null value to another value.
The data types whose can be used are, date, character and number.

Syntax:

NVL (expr1, expr2)

The expr1 is the source values and the expr2 is the target value for converting the null.

Data Type	Conversion Example
NUMBER	NVL(number_column,9)
DATE	NVL(date_column,'01-JAN-95')
CHAR or VARCHAR2	NVL(character_column,'Null value')

```
SQL> SELECT ename,NVL(TO_CHAR(mgr), 'No manager')
2 FROM emp;

      ENAME      NVL (TO_CHAR (MGR) , 'NOMANAGER')
-----
SMITH          7902
ALLEN          7698
WARD           7698
JONES          7839
MARTIN         7698
BLAKE          7839
CLARK          7839
SCOTT          7566
KING           No manager
TURNER         7698
...
17 row(s) selected.
```

4.7.2.Using the NVL2 function

The **NVL2** function examines the first expression. If the first expression is not null, the **NVL2** function return the second expression, else the third expression is returned.

Syntax:

NVL2 (*expr1*, *expr2*, *expr3*)

Expr1 is the source values, *expr2* is the value returned if the *expr1* value is null , and *expr3* is the value returned if the *expr2* is null.

```
SQL> SELECT ename,sal,comm,NVL2(comm,'SAL+COM','SAL') income
2  FROM emp;
```

ENAME	SAL	COMM	INCOME
SMITH	800	3000	SAL+COM
ALLEN	1600	500	SAL+COM
WARD	1250	500	SAL+COM
JONES	2975		SAL
MARTIN	1250	1400	SAL+COM
BLAKE	2850		SAL
...			

17 row(s) selected.

4.7.3.Using the NULLIF function

The function compares two expressions. If they are equal the function returns null else the function returns the first value. You cannot specify the literal NULL for first expression.

Syntax:**NULLIF** (*expr1*, *expr2*)

Expr1 is the source value compared with *expr2*. *Expr2* is the source value compared with *expr1*.

```
SQL> SELECT ename,LENGTH(ename) "Expr1",job,LENGTH(job) "Expr2",
2  NULLIF(LENGTH(firstname),LENGTH(job)) "Result"
3  FROM emp;
NULLIF(LENGTH(ename),LENGTH(job)) "Result";
```

ENAME	Expr1	JOB	Expr2	Result
SMITH	5	CLERK	5	
ALLEN	5	SALESMAN	8	5
WARD	4	SALESMAN	8	4
JONES	5	MANAGER	7	5
MARTIN	6	SALESMAN	8	6
BLAKE	5	MANAGER	7	5
...				

17 row(s) selected.

4.7.4.Using the COALESCE function

The **COALESCE** function can take multiple alternate values unlike to **NVL**. If the fist expression is not null , it returns that expression, else it tests the next expression, and returns the first non-null expression in the list.

Syntax:**COALESCE** (*expr1*, *expr2*, ..., *exprn*)

The first non-null expression in the list is returned.

The employee COALESCE has no salary, no commission and no manager, so the **COALESCE** function returns the first non-null value in the list which is 10.

```
SQL> SELECT ename,COALESCE(comm,mgr,sal,10)
2  FROM emp;

      ENAME      COALESCE(COMM,MGR,SAL,10)
-----
SMITH              3000
ALLEN              500
WARD              500
JONES            7839
MARTIN            1400
BLAKE            7839
CLARK            7839
SCOTT            7566
KING             5000
TURNER           0
COALESCE         10
...
17 row(s) selected.
```

4.8.Conditional expressions

4.8.1.Using the CASE expression

The **CASE** expression, allows you to use the *IF-THEN-ELSE* logic in SQL statements, without having to invoke procedures.

When using a **CASE** expression, Oracle looks for the first **WHEN...THEN** pair for which the first expression is equal to the second one and returns a specified value.

If none of the **WHEN...THEN** pairs meet this condition and an **ELSE** clause exists, the expression returns a specified value for this situation.

If there is no **ELSE** clause the **CASE** expression returns NULL.

All the expressions must be of the same data.

In this SQL statement, the value of job is decoded.

According to the job , the employee salaries are increased.

If job is CLERK the increase is 10 percent, if job is SALESMAN the increase is 20 percent, if job is president the increase is 200 percent.

For all other job there is no increase.

```
SQL> SELECT ename,job,sal,
2  CASE job WHEN 'CLERK' THEN 1.10*sal
3  WHEN 'PRESIDENT' THEN 2.0*sal
4  WHEN 'SALESMAN' THEN 1.20*sal
5  ELSE sal END "Revised salary"
6  FROM emp;

      ENAME      JOB      SAL Revised salary
-----
SMITH      CLERK      800      880
ALLEN      SALESMAN    1600    1920
WARD      SALESMAN    1250    1500
JONES      MANAGER     2975    2975
MARTIN      SALESMAN    1250    1500
BLAKE      MANAGER     2850    2850
CLARK      MANAGER     2450    2450
...
17 row(s) selected.
```

4.8.2.Using the DECODE function

The **DECODE** function decodes expression in a way similar to the **COALESCE**.

It decodes an expression after comparing it to a specified other expression.

If the two expressions are equal a specified value is returned, else the **DECODE** function returns NULL.

According to the job , the employee salaries are increased.

If job is CLERK the increase is 10 percent, is job is SALESMAN the increase is 20 percent, if job is president the increase is 200 percent.

For all other job there is no increase

```
SQL> SELECT ename,job,sal,
2  DECODE (job , 'CLERK',      1.10*sal,
3           , 'PRESIDENT', 2.0*sal,
4           , 'SALESMAN',  1.20*sal,
5           sal) "Revised salary"
6  FROM emp;
```

ENAME	JOB	SAL	Revised salary
SMITH	CLERK	800	880
ALLEN	SALESMAN	1600	1920
WARD	SALESMAN	1250	1500
JONES	MANAGER	2975	2975
MARTIN	SALESMAN	1250	1500
BLAKE	MANAGER	2850	2850
CLARK	MANAGER	2450	2450
...			

17 row(s) selected.