Introduction to Database

Lecture Notes Midterm

American International University-Bangladesh

Lecture Notes (Lecture01_IDB)

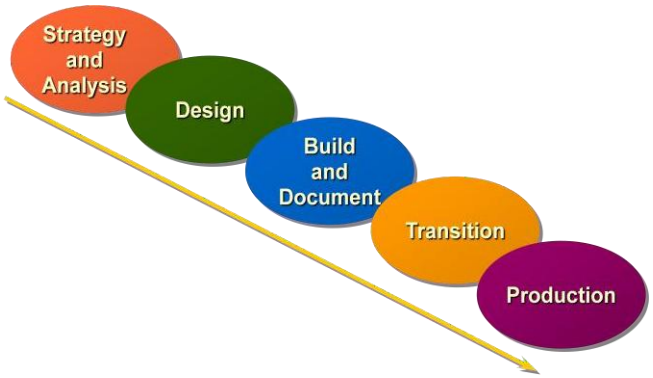
System Requirement:

1. Operating System- Windows XP/7/8/NT
2. Software- Oracle 9i/10g/11g/xe

Objectives:

- Discuss the theoretical and physical aspects of a relational database
- Describe the Oracle implementation of the RDBMS and ORDBMS
- Describe how SQL and PL/SQL are used in the Oracle product set
- Describe the use and benefits of PL/SQL

System Development Life Cycle



System Development Life Cycle

From concept to production, you can develop a database by using the system development life cycle, which contains multiple stages of development. This top-down, systematic approach to database development transforms business information requirements into an operational database.

Strategy and Analysis

- Study and analyze the business requirements. Interview users and managers to identify the information requirements. Incorporate the enterprise and application mission statements as well as any future system specifications.

- Build models of the system. Transfer the business narrative into a graphical representation of business information needs and rules. Confirm and refine the model with the analysts and experts.

Design

Design the database based on the model developed in the strategy and analysis phase.

Build and Document

- Build the prototype system. Write and execute the commands to create the tables and supporting objects for the database.
- Develop user documentation, help text, and operations manuals to support the use and operation of the system.

Transition

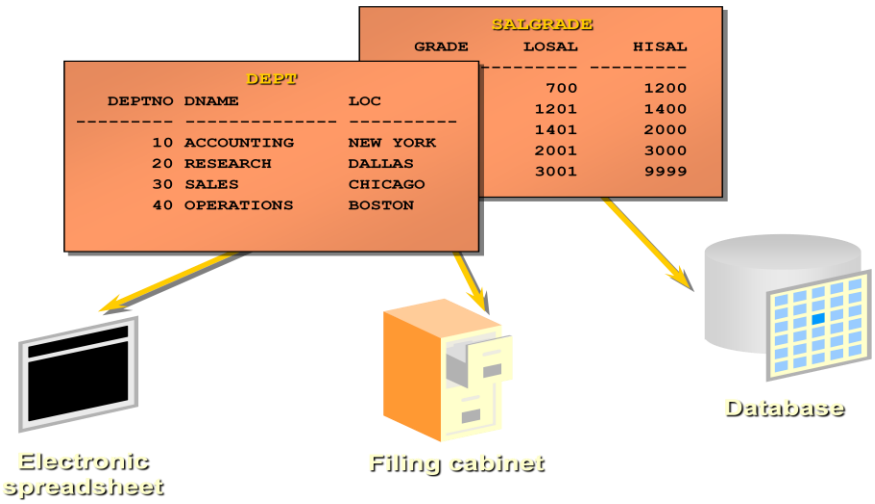
Refine the prototype. Move an application into production with user acceptance testing, conversion of existing data, and parallel operations. Make any modifications required.

Production

Roll out the system to the users. Operate the production system. Monitor its performance, and enhance and refine the system.

Note: The various phases of system development life cycle can be carried out iteratively. This course focuses on the build phase of the system development life cycle.

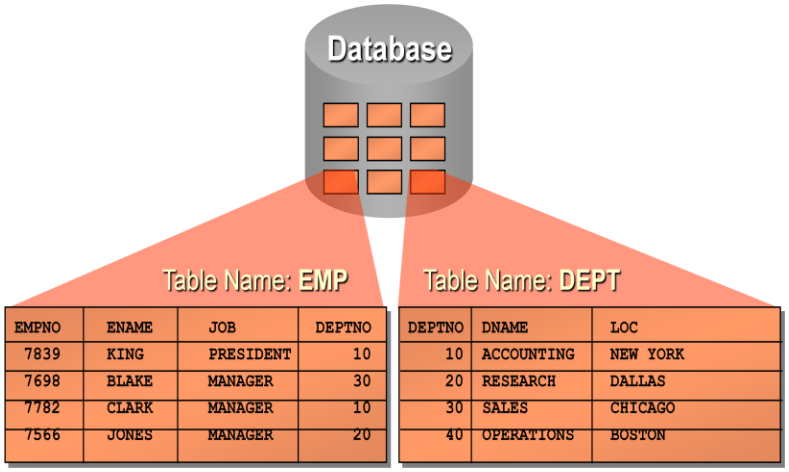
Data Storage on Different Media



Relational Database Concept

- Dr. E. F. Codd proposed the relational model for database systems in 1970.
- It is the basis for the relational database management system (RDBMS).
- The relational model consists of the following:
- Collection of objects or relations
- Set of operators to act on the relations
- Data integrity for accuracy and consistency

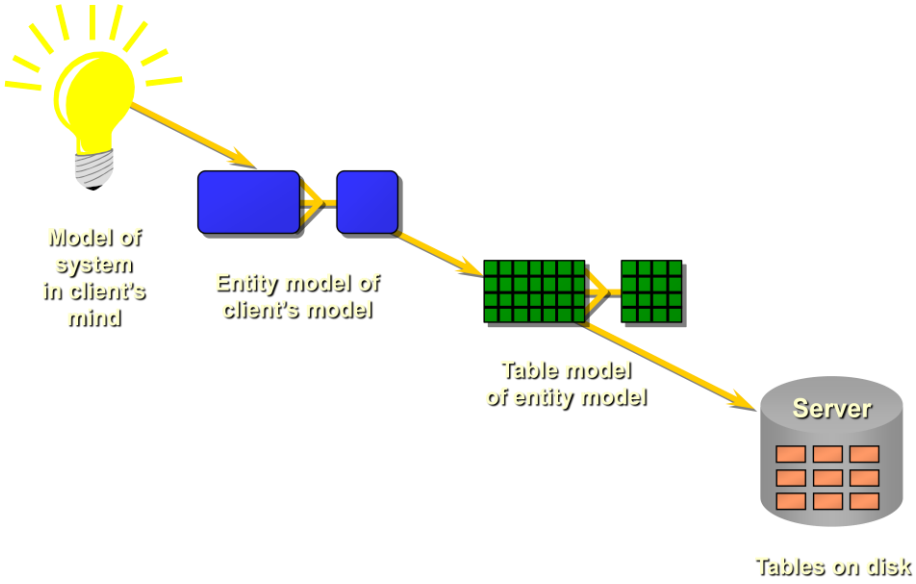
A relational database is a collection of relations or two-dimensional tables.



EMPNO	ENAME	JOB	DEPTNO
7839	KING	PRESIDENT	10
7698	BLAKE	MANAGER	30
7782	CLARK	MANAGER	10
7566	JONES	MANAGER	20

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

Data Models



Entity Relationship Model

ER Modeling

In an effective system, data is divided into discrete categories or entities. An *entity relationship (ER)* model is an illustration of various entities in a business and the relationships between them. An ER model is derived from business specifications or narratives and built during the analysis phase of the system development life cycle. ER models separate the information required by a business from the activities performed within a business. Although businesses can change their activities, the type of information tends to remain constant. Therefore, the data structures also tend to be constant.

Benefits of ER Modeling

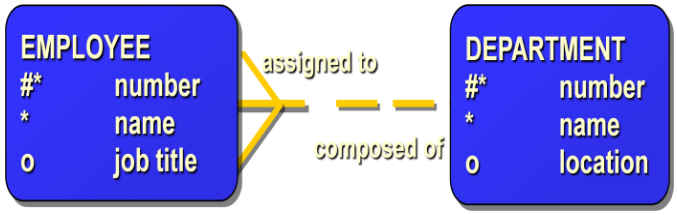
- Documents information for the organization in a clear, precise format
- Provides a clear picture of the scope of the information requirement
- Provides an easily understood pictorial map for the database design
- Offers an effective framework for integrating multiple applications

Key Components

- Entity: A thing of significance about which information needs to be known. Examples are departments, employees, and orders.
- Attribute: Something that describes or qualifies an entity. For example, for the employee entity, the attributes would be the employee number, name, job title, hire date, department number, and so on. Each of the attributes is either required or optional. This state is called *optionality*.
- Relationship: A named association between entities showing optionality and degree. Examples are employees and departments, and orders and items.

Create an entity relationship diagram from business specifications or narratives
Scenario

- ". . . Assign one or more employees to a department . . ."
- ". . . Some departments do not yet have assigned employees . . ."



Entities

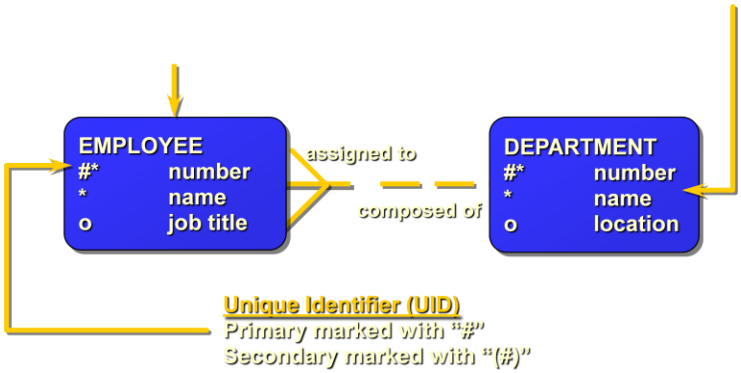
To represent an entity in a model, use the following conventions:

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

Attributes

To represent an attribute in a model, use the following conventions:

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



Relationships

Each direction of the relationship contains:

- A name, for example, *taught by* or *assigned to*
- An optionality, either *must be* or *may be*
- A degree, either *one and only one* or *one or more*

Note: The term *cardinality* is a synonym for the term *degree*.

Each source entity {may be | must be} relationship name {one and only one | one or more} destination entity.

Note: The convention is to read clockwise.

Unique Identifiers

A unique identifier (UID) is any combination of attributes or relationships, or both, that serves to distinguish occurrences of an entity. Each entity occurrence must be uniquely identifiable.

- Tag each attribute that is part of the UID with a number symbol: #
- Tag secondary UIDs with a number sign in parentheses: (#)

Relating Multiple Tables

Each table contains data that describes exactly one entity. For example, the EMP table contains information about employees. Categories of data are listed across the top of each table, and individual cases are listed below. Using a table format, you can readily visualize, understand, and use information.

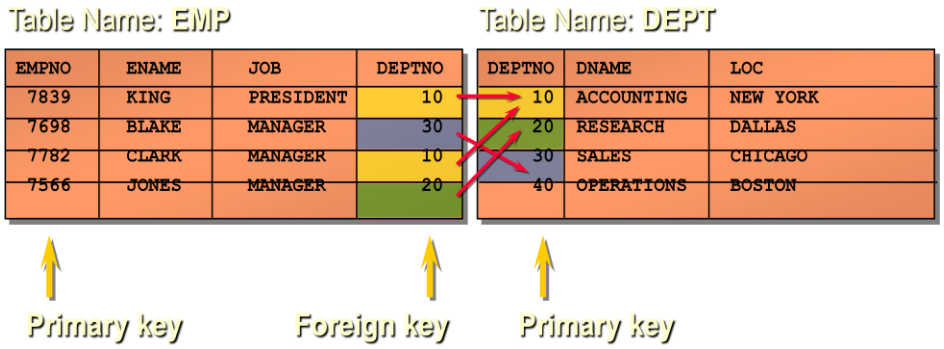
Because data about different entities is stored in different tables, you may need to combine two or more tables to answer a particular question. For example, you may want to know the location of the department where an employee works. In this scenario, you need information from the EMP table (which contains data about employees) and the DEPT table (which contains information about departments). An RDBMS enables you to relate the data in one table to the data in another by using the foreign keys. A foreign key is a column or a set of columns that refer to a primary key in the same table or another table.

The ability to relate data in one table to data in another enables you to organize information in separate, manageable units. Employee data can be kept logically distinct from department data by storing it in a separate table.

Guidelines for Primary Keys and Foreign Keys

- No duplicate values are allowed 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.



Properties of a Relational Database

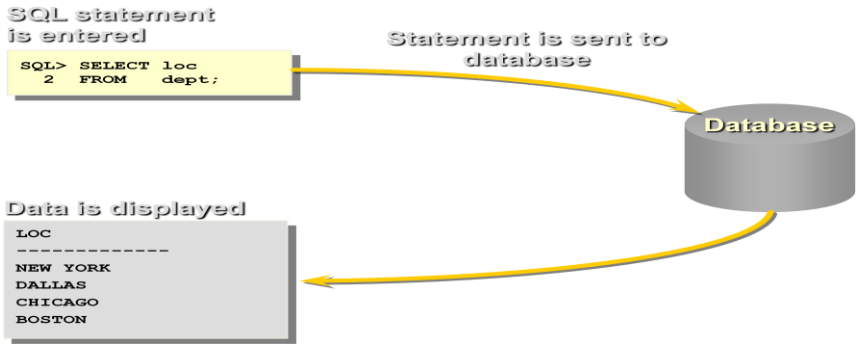
In a relational database, you do not specify the access route to the tables, and you do not need to know how the data is arranged physically.

To access the database, you execute a structured query language (SQL) statement, which is the American National Standards Institute (ANSI) standard language for operating upon relational databases. The language contains a large set of operators for partitioning and combining relations. The database can be modified by using the SQL statements.

Structured Query Language

SQL allows you to communicate with the server and has the following advantages:

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

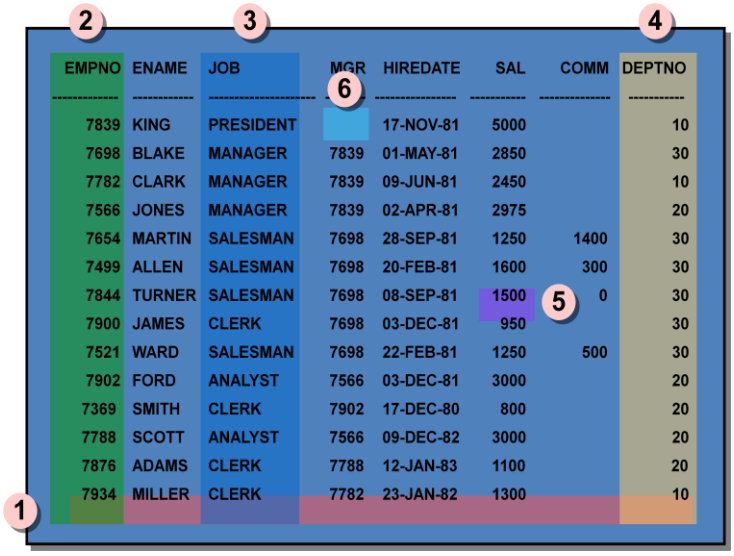


Terminology Used in a Relational Database

A relational database can contain one or many tables. A *table* is the basic storage structure of an RDBMS. A table holds all the data necessary about something in the real world—for example, employees, invoices, or customers.

The image below shows the contents of the EMP *table* or *relation.* The numbers indicate the following:

1. A single *row* or *tuple* representing all data required for a particular employee. Each row in a table should be identified by a primary key, which allows no duplicate rows. The order of rows is insignificant; specify the row order when the data is retrieved.
2. A *column* or *attribute* containing the employee number, which is also the primary key. The employee number identifies a *unique* employee in the EMP table. A primary key must contain a value.
3. A column that is not a key value. A column represents one kind of data in a table; in the example, the job title of all the employees. Column order is insignificant when storing data; specify the column order when the data is retrieved.



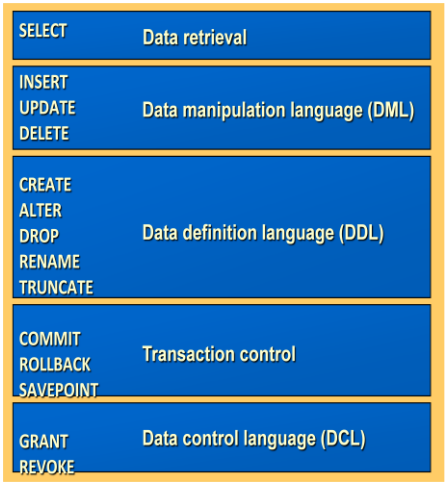
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		17-NOV-81	5000		10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600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

4. A column containing the department number, which is also a *foreign key*. A foreign key is a column that defines how tables relate to each other. A foreign key refers to a primary key or a unique key in another table. In the example, DEPTNO *uniquely* identifies a department in the DEPT table.
5. A *field* can be found at the intersection of a row and a column. There can be only one value in it.
6. A field may have no value in it. This is called a *null value*. In the EMP table, only employees who have a role of salesman have a value in the COMM (commission) field.

Note: Null values are covered further in subsequent lessons.

SQL Statements

Oracle SQL complies with industry-accepted standards. Oracle Corporation ensures future compliance with evolving standards by actively involving key personnel in SQL standards committees. Industry-accepted committees are the American National Standards Institute (ANSI) and the International Standards Organization (ISO). Both ANSI and ISO have accepted SQL as the standard language for relational databases.



About PL/SQL

Procedural Language/SQL (PL/SQL) is Oracle Corporation's procedural language extension to SQL, the standard data access language for object-relational databases. PL/SQL offers modern software engineering features such as data encapsulation, exception handling, information hiding, and object orientation, and so brings state-of-the-art programming to the Oracle Server and Toolset.

PL/SQL incorporates many of the advanced features made in programming languages designed during the 1970s and 1980s. It allows the data manipulation and query statements of SQL to be included in block-structured and procedural units of code, making PL/SQL a powerful transaction processing language. With PL/SQL, you can use SQL statements to finesse Oracle data and PL/SQL control statements to process the data.

Tables Used in the Course

The following three tables will be used in this course:

- EMP table, which gives details of all the employees
- DEPT table, which gives details of all the departments
- SALGRADE table, which gives details of salaries for various grades

Lecture Notes (Lecture02_IDB)

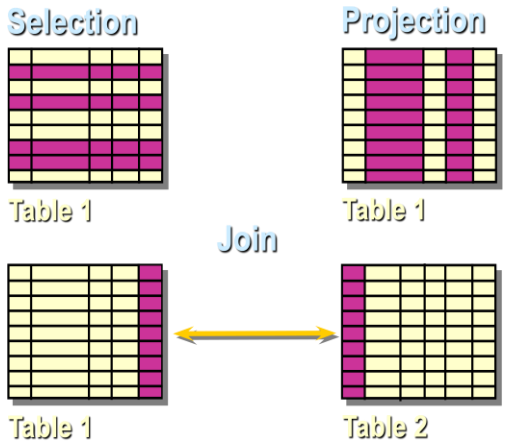
Objectives:

To extract data from the database, you need to use the structured query language (SQL) SELECT statement. You may need to restrict the columns that are displayed. This lesson describes all the SQL statements that you need to perform these actions. You may want to create SELECT statements that can be used time and time again. This lesson also covers the use of SQL*Plus commands to execute SQL statements.

Capabilities of SQL SELECT Statements

A SELECT statement retrieves information from the database. Using a SELECT statement, you can do the following:

- Selection: You can use the selection capability in SQL to choose the rows in a table that you want returned by a query. You can use various criteria to selectively restrict the rows that you see.
- Projection: You can use the projection capability in SQL to choose the columns in a table that you want returned by your query. You can choose as few or as many columns of the table as you require.
- Join: You can use the join capability in SQL to bring together data that is stored in different tables by creating a link through a column that both the tables share. You will learn more about joins in a later lesson.



Basic SELECT Statement

In its simplest form, a SELECT statement must include the following:

A SELECT clause, which specifies the columns to be displayed

A FROM clause, which specifies the table containing the columns listed in the SELECT clause

In the syntax:

SELECT-is a list of one or more columns.
DISTINCT- suppresses duplicates.
****-*** selects all columns
Column- selects the named column.
Alias gives selected columns different headings.
FROM *table* specifies the table containing the columns.

Note: Throughout this course, the words keyword, clause, and statement are used.

A *keyword* refers to an individual SQL element.
For example, SELECT and FROM are keywords.
A *clause* is a part of an SQL statement.
For example, SELECT empno, ename, … is a clause.
A *statement* is a combination of two or more clauses.
For example, SELECT * FROM emp is a SQL statement.

Writing SQL Statements

Using the following simple rules and guidelines, you can construct valid statements that are both easy to read and easy to edit:

• SQL statements are not case sensitive, unless indicated. SQL statements can be entered on one or many lines.

• Keywords cannot be split across lines or abbreviated. Clauses are usually placed on separate lines for readability and ease of editing.

• Tabs and indents can be used to make code more readable. Keywords typically are entered in uppercase; all other words, such as table names and columns, are entered in lowercase.

• Within SQL*Plus, a SQL statement is entered at the SQL prompt, and the subsequent lines are numbered. This is called the *SQL buffer*. Only one statement can be current at any time within the buffer.

Executing SQL Statements

- Place a semicolon (;) at the end of the last clause.
- Place a slash on the last line in the buffer.
- Place a slash at the SQL prompt.

Issue a SQL*Plus RUN command at the SQL prompt.

Selecting All Columns, All Rows

You can display *all* columns of data in a table by following the SELECT keyword with an asterisk (*). In the example below, the department table contains three columns: DEPTNO, DNAME, and LOC. The table contains four rows, one for each department.

You can also display *all* columns in the table by listing all the columns after the SELECT keyword. For example, the following SQL statement, like the example shown below, displays all columns and all rows of the DEPT table:

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

```
    DEPTNO LOC
---------- -------------
        10 NEW YORK
        20 DALLAS
        30 CHICAGO
        40 BOSTON
```

Character column heading and data as well as date column heading and data are left-justified within a column width. Number headings and data are right-justified.

Arithmetic Expressions

You may need to modify the way in which data is displayed, perform calculations, or look at what-if scenarios. This is possible using arithmetic expressions. An arithmetic expression may contain column names, constant numeric values, and the arithmetic operators.

Arithmetic Operators

The list shows the arithmetic operators available in SQL. You can use arithmetic operators in any clause of a SQL statement except the FROM clause.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Using Arithmetic Operators

The example uses the addition operator to calculate a salary increase of $300 for all employees and displays a new SAL+300 column in the output. Note that the resultant calculated column SAL+300 is not a new column in the EMP table; it is for display only. By default, the name of a new column comes from the calculation that generated it—in this case, sal+300.

Note: SQL*Plus ignores blank spaces before and after the arithmetic operator.

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

Operator Precedence

If an arithmetic expression contains more than one operator, multiplication and division are evaluated first. If operators within an expression are of same priority, then evaluation is done from left to right. You can use parentheses to force the expression within parentheses to be evaluated first. The example displays the name, salary, and annual compensation of employees. It calculates the annual compensation as 12 multiplied by the monthly salary, plus a one-time bonus of $100. Notice that multiplication is performed before addition.

Note: Use parentheses to reinforce the standard order of precedence and to improve clarity. For example, the expression above can be written as (12*sal)+100 with no change in the result.

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

ENAME            SAL 12*SAL+100
---------- --------- ----------
KING            5000      60100
BLAKE           2850      34300
CLARK           2450      29500
JONES           2975      35800
MARTIN          1250      15100
ALLEN           1600      19300
...
14 rows selected.
```

Using Parentheses

You can override the rules of precedence by using *parentheses* to specify the order in which operators are executed. The example displays the name, salary, and annual compensation of employees. It calculates the annual compensation as monthly salary plus a monthly bonus of $100, multiplied by 12. Because of the parentheses, addition takes priority over multiplication.

Null Values

If a row lacks the data value for a particular column, that value is said to be *null*, or to contain null. A null value is a value that is unavailable, unassigned, unknown, or inapplicable. A null value is not the same as zero or a space. Zero is a number, and a space is a character. Columns of any datatype can contain null values, unless the column was defined as NOT NULL or as PRIMARY KEY when the column was created.

In the COMM column in the EMP table, you notice that only a SALESMAN can earn commission. Other employees are not entitled to earn commission. A null value represents that fact. Turner, who is a salesman, does not earn any commission. Notice that his commission is zero and not null. If any column value in an arithmetic expression is null, the result is null. For example, if you attempt to perform division with zero, you get an error. However, if you divide a number by null, the result is a null or unknown. In the example given below, employee KING is not in SALESMAN and does not get any commission. Because the COMM column in the arithmetic expression is null, the result is null. For more information, see *Oracle Server SQL Reference,* Release 8, and “Elements of SQL.”

```
SQL> SELECT   ename, job, comm
  2  FROM     emp;
```

```
ENAME      JOB          COMM
---------- --------- ---------
KING       PRESIDENT
BLAKE      MANAGER
...
TURNER     SALESMAN          0
...
14 rows selected.
```

Column Aliases

When displaying the result of a query, SQL*Plus normally uses the name of the selected column as the column heading. In many cases, this heading may not be descriptive and hence is difficult to understand. You can change a column heading by using a column alias. Specify the alias after the column in the SELECT list using a space as a separator. By default, alias headings appear in uppercase. If the alias contains spaces, special characters (such as # or $), or is case sensitive, enclose the alias in double quotation marks (“ “).

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

```
NAME            SALARY
------------- ---------
...
```

```
SQL> SELECT ename "Name",
  2         sal*12 "Annual Salary"
  3  FROM   emp;
```

```
Name          Annual Salary
------------- -------------
...
```

Concatenation Operator

You can link columns to other columns, arithmetic expressions, or constant values to create a character expression by using the concatenation operator (||). Columns on either side of the operator are combined to make a single output column.

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

```
Employees
------------------
KINGPRESIDENT
BLAKEMANAGER
CLARKMANAGER
JONESMANAGER
MARTINSALESMAN
ALLENSALESMAN
...
14 rows selected.
```

Literal Character Strings

A literal is any character, expression, or number included in the SELECT list that is not a column name or a column alias. It is printed for each row returned. Literal strings of free-format text can be included in the query result and are treated the same as a column in the SELECT list. Date and character literals *must* be enclosed within single quotation marks (‘ ‘); number literals must not. The below example shows the names and jobs of all employees. The column has the heading Employee Details. Notice the spaces between the single quotation marks in the SELECT statement. The spaces improve the readability of the output. In the following example, the name and salary for each employee is concatenated with a literal to give the returned rows more meaning.

```
SQL> SELECT ename ||' '||'is a'||' '||job
  2               AS "Employee Details"
  3  FROM   emp;
```

```
Employee Details
------------------------
KING is a PRESIDENT
BLAKE is a MANAGER
CLARK is a MANAGER
JONES is a MANAGER
MARTIN is a SALESMAN
...
14 rows selected.
```

Duplicate Rows

Unless you indicate otherwise, SQL*Plus displays the results of a query without eliminating duplicate rows. The example below displays all the department numbers from the EMP table. Notice that the department numbers are repeated. To eliminate duplicate rows in the result, include the DISTINCT keyword in the SELECT clause immediately after the SELECT keyword. In the example, the EMP table actually contains fourteen rows but there are only three unique department numbers in the table. You can specify multiple columns after the DISTINCT qualifier. The DISTINCT qualifier affects all the selected columns, and the result represents a distinct combination of the columns.

```
SQL> SELECT DISTINCT deptno
  2  FROM   emp;
```

```
    DEPTNO
---------
        10
        20
        30
```

Displaying Table Structure

In SQL*Plus, you can display the structure of a table using the DESCRIBE command. The result of the command is to see the column names and datatypes as well as whether a column *must* contain data.

In the syntax:
tablename is the name of any existing table, view, or synonym accessible to the user. The example displays the information about the structure of the DEPT table. The datatypes are described in the following table:

```
SQL> DESCRIBE dept
```

```
Name                 Null?    Type
-------------------- -------- ------------
DEPTNO               NOT NULL NUMBER(2)
DNAME                         VARCHAR2(14)
LOC                           VARCHAR2(13)
```

In the result:

Null?	Indicates whether a column *must* contain data; NOT NULL indicates that a column must contain data
Type	displays the datatype for a column

Exercise:

1. Write a query to display the name, department number, and department name for all employees.
2. Create a unique listing of all jobs that are in department 30. Include the location of department 30 in the output.
3. Write a query to display the employee name, department name, and location of all employees who earn a commission.
   Display the employee name and department name for all employees who have an A in their name.

Lecture Notes (Lecture03_IDB)

- For details go through Database System Concepts, 6th Edition, Chapter 01

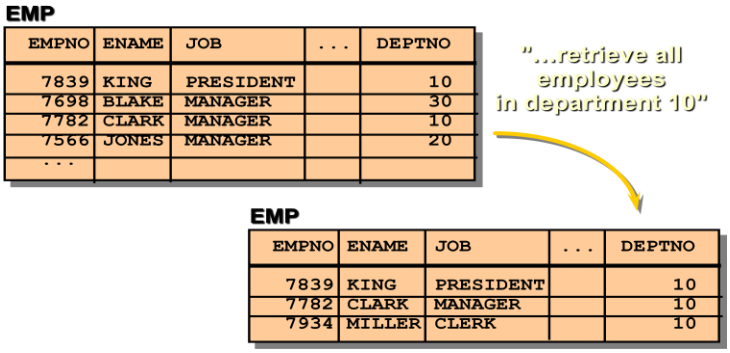
Lecture Notes (Lecture04_IDB)

Objectives:

While retrieving data from the database, you may need to restrict the rows of data that are displayed or specify the order in which the rows are displayed. This lesson explains the SQL statements that you will use to perform these actions.

Limiting Rows Using a Selection

In the example stated below, assume that you want to display all the employees in department 10. The highlighted set of rows with a value of 10 in DEPTNO column are the only ones returned. This method of restriction is the basis of the WHERE clause in SQL.



Limiting Rows Selected

You can restrict the rows returned from the query by using the WHERE clause. A 'WHERE clause contains a condition that must be met and it directly follows the FROM clause. In the syntax:

WHERE	restricts the query to rows that meet a condition
Condition	is composed of column names, expressions, constants, and a comparison operator

The WHERE clause can compare values in columns, literal values, arithmetic expressions or functions. The WHERE clause consists of three elements:

- Column name
- Comparison operator
- Column name, constant, or list of values

```
SELECT      [DISTINCT] {*| column [alias], ...}
FROM        table
[WHERE      condition(s)];
```

Using the WHERE clause

In the example, the SELECT statement retrieves the name, job title, and department number of all employees whose job title is CLERK. Note that the job title CLERK has been specified in uppercase to ensure that the match is made with the job column in the EMP table. Character strings are case sensitive.

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

```
ENAME      JOB          DEPTNO
---------- --------- ---------
JAMES      CLERK            30
SMITH      CLERK            20
ADAMS      CLERK            20
MILLER     CLERK            10
```

Character Strings and Dates

Character strings and dates in the WHERE clause must be enclosed in single quotation marks (''). Number constants, however, should not. All character searches are case sensitive. In the following example, no rows are returned because the EMP table stores all the data in uppercase:

SQL> SELECT ename, empno, job, deptno FROM emp WHERE job='clerk';

Oracle stores dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds. The default date display is DD-MON-YY.

Comparison Operators

Comparison operators are used in conditions that compare one expression to another. They are used in the WHERE clause in the following format:

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

Syntax

… WHERE *expr operator value*

Examples

… WHERE hiredate='01-JAN-95'
… WHERE sal>=1500
… WHERE ename='SMITH'

Using the Comparison Operators

In the example, the SELECT statement retrieves name, salary, and commission from the EMP table, where the employee salary is less than or equal to the commission amount. Note that there is no explicit value supplied to the WHERE clause. The two values being compared are taken from the SAL and COMM columns in the EMP table.

```
SQL> SELECT ename, sal, comm
  2  FROM   emp
  3  WHERE  sal<=comm;
```

```
ENAME          SAL       COMM
---------- --------- ---------
MARTIN        1250 <--> 1400
```

Other Comparison Operators

Operator	Meaning
BETWEEN ...AND...	Between two values (inclusive)
IN(list)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

The BETWEEN Operator

You can display rows based on a range of values using the BETWEEN operator. The range that you specify contains a lower range and an upper range. The SELECT statement given below returns rows from the EMP table for any employee whose salary is between $1000 and $1500. Values specified with the BETWEEN operator are inclusive. You must specify the lower limit first.

```
SQL> SELECT   ename, sal
  2  FROM     emp
  3  WHERE    sal BETWEEN 1000 AND 1500;
```

The IN Operator

To test for values in a specified list you have to use the IN operator. The example displays employee number, name, salary, and manager's employee number of all the employees whose manager's employee number is 7902, 7566, or 7788. The IN operator can be used with any datatype. The following example returns a row from the EMP table for any employee whose name is included in the list of names in the WHERE clause:

SQL> SELECT empno, ename, mgr, deptno FROM emp WHERE ename IN ('FORD' , 'ALLEN');

If characters or dates are used in the list, they must be enclosed in single quotation marks (‘’).

The LIKE Operator

You may not always know the exact value to search for. You can select rows that match a character pattern by using the LIKE operator. The character pattern-matching operation is referred to as a *wildcard* search. Two symbols can be used to construct the search string. The SELECT statement above returns the employee name from the EMP table for any employee whose name begins with an "S." Note the uppercase "S." Names beginning with an "s" will not be returned. The LIKE operator can be used as a shortcut for some BETWEEN comparisons. The following example displays the names and hire dates of all employees who joined between January 1981 and December 1981:

SQL> SELECT ename, hiredate FROM emp WHERE hiredate LIKE '%81';

Combining Wildcard Characters

The % and _ symbols can be used in any combination with literal characters. The example displays the names of all employees whose name has an "A" as the second character.

The ESCAPE Option

When you need to have an exact match for the actual '%' and '_' characters, use the ESCAPE option. This option specifies what the ESCAPE character is. To display the names of employees whose name contains 'A_B', use the following SQL statement:

```
SQL> SELECT   ename
  2  FROM     emp
  3  WHERE    ename LIKE '_A%';
```

```
ENAME
----------
MARTIN
JAMES
WARD
```

The ESCAPE option identifies the backslash (\) as the escape character. In the pattern, the escape character precedes the underscore (_). This causes the Oracle Server to interpret the underscore literally.

The IS NULL Operator

The IS NULL operator tests for values that are null. A null value means the value is unavailable, unassigned, unknown, or inapplicable. Therefore, you cannot test with (=) because a null value cannot be equal or unequal to any value. The example retrieves the name and manager of all employees who do not have a manager. For example, to display name, job title, and commission for all employees who are not entitled to get a commission, use the following SQL statement:

```
SQL> SELECT  ename, mgr
  2  FROM    emp
  3  WHERE   mgr IS NULL;
```

Logical Operators

A logical operator combines the result of two component conditions to produce a single result based on them or to invert the result of a single condition. Three logical operators are available in SQL:

- AND
- OR
- NOT

All the examples so far have specified only one condition in the WHERE clause. You can use several conditions in one WHERE clause using the AND and OR operators.

Operator	Meaning
AND	Returns TRUE if *both* component conditions are TRUE
OR	Returns TRUE if *either* component condition is TRUE
NOT	Returns TRUE if the following condition is FALSE

```
SQL> SELECT empno, ename, job, sal
  2  FROM   emp
  3  WHERE  sal>=1100
  4  AND    job='CLERK';
```

```
    EMPNO ENAME      JOB              SAL
--------- ---------- --------- ---------
     7876 ADAMS      CLERK            1100
     7934 MILLER     CLERK            1300
```

```
SQL> SELECT empno, ename, job, sal
  2  FROM   emp
  3  WHERE  sal>=1100
  4  OR     job='CLERK';
```

```
    EMPNO ENAME      JOB          SAL
--------- ---------- --------- ---------
     7839 KING       PRESIDENT      5000
     7698 BLAKE      MANAGER        2850
     7782 CLARK      MANAGER        2450
     7566 JONES      MANAGER        2975
     7654 MARTIN     SALESMAN       1250
     ...
     7900 JAMES      CLERK           950
     ...
14 rows selected.
```

```
SQL> SELECT ename, job
  2  FROM   emp
  3  WHERE  job NOT IN ('CLERK','MANAGER','ANALYST');
```

```
ENAME      JOB
---------- ---------
KING       PRESIDENT
MARTIN     SALESMAN
ALLEN      SALESMAN
TURNER     SALESMAN
WARD       SALESMAN
```

Rules of Precedence

Order Evaluated	Operator
1	All comparison operators
2	NOT
3	AND
4	OR

```
SQL> SELECT ename, job, sal
  2  FROM   emp
  3  WHERE  job='SALESMAN'
  4  OR     job='PRESIDENT'
  5  AND    sal>1500;
```

```
ENAME      JOB              SAL
---------- --------- ---------
KING       PRESIDENT       5000
MARTIN     SALESMAN        1250
ALLEN      SALESMAN        1600
TURNER     SALESMAN        1500
WARD       SALESMAN        1250
```

The ORDER BY Clause

The order of rows returned in a query result is undefined. The ORDER BY clause can be used to sort the rows. If you use the ORDER BY clause, you must place last. You can specify an expression or an alias to sort.

Syntax

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

Where:

ORDER BY specifies the order in which the retrieved rows are displayed
ASC orders the rows in ascending order (this is the default order)
DESC orders the rows in descending order

If the ORDER BY clause is not used, the sort order is undefined, and the Oracle Server may not fetch rows in the same order for the same query twice. Use the ORDER BY clause to display the rows in a specific order.

```
SQL> SELECT   ename, job, deptno, hiredate
  2  FROM     emp
  3  ORDER BY hiredate;

ENAME      JOB          DEPTNO HIREDATE
---------- --------- --------- ---------
SMITH      CLERK            20 17-DEC-80
ALLEN      SALESMAN         30 20-FEB-81
...
14 rows selected.
```

Default Ordering of Data

The default sort order is ascending:

- Numeric values are displayed with the lowest values first—for example, 1–999.
- Date values are displayed with the earliest value first—for example, 01-JAN-92 before 01-JAN-95.
- Character values are displayed in alphabetical order—for example, A first and Z last.
- Null values are displayed last for ascending sequences and first for descending sequences.

Reversing the Default Order

To reverse the order in which rows are displayed, specify the keyword DESC after the column name in the ORDER BY clause. The example sorts the result by the most recently hired employee.

```
SQL> SELECT    ename, job, deptno, hiredate
  2  FROM      emp
  3  ORDER BY hiredate DESC;
```

```
ENAME      JOB          DEPTNO HIREDATE
---------- --------- --------- ---------
ADAMS      CLERK            20 12-JAN-83
SCOTT      ANALYST          20 09-DEC-82
MILLER     CLERK            10 23-JAN-82
JAMES      CLERK            30 03-DEC-81
FORD       ANALYST          20 03-DEC-81
KING       PRESIDENT        10 17-NOV-81
MARTIN     SALESMAN         30 28-SEP-81
...
14 rows selected.
```

Sorting by Multiple Columns

You can sort query results by more than one column. The sort limit is the number of columns in the given table. In the ORDER BY clause, specify the columns, and separate the column names using commas. If you want to reverse the order of a column, specify DESC after its name. You can order by columns that are not included in the SELECT clause.

Example

Display name and salary of all employees. Order the result by department number and then descending order by salary.

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

```
ENAME        DEPTNO        SAL
---------- --------- ---------
KING             10       5000
CLARK            10       2450
MILLER           10       1300
FORD             20       3000
...
14 rows selected.
```

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

Exercise:

1. Create a query to display the name and salary of employees earning more than $2850.

ENAME	SAL
KING	5000
JONES	2975
FORD	3000
SCOTT	3000

2. Create a query to display the employee name and department number for employee number 7566.

ENAME	DEPTNO
JONES	20

3. Display the employee name, job, and start date of employees hired between February 20, 1981, and May 1, 1981. Order the query in ascending order by start date.

ENAME	JOB	HIREDATE
ALLEN	SALESMAN	20-FEB-81
WARD	SALESMAN	22-FEB-81
JONES	MANAGER	02-APR-81
BLAKE	MANAGER	01-MAY-81

5. Display the employee name and department number of all employees in departments 10 and 30 in alphabetical order by name.

ENAME	DEPTNO
ALLEN	30
BLAKE	30
CLARK	10
JAMES	30
KING	10
MARTIN	30
MILLER	10
TURNER	30
WARD	30

6. Write a query to list the name and salary of employees who earn more than $1500 and are in department 10 or 30. Label the columns Employee and Monthly Salary, respectively. Resave your SQL statement to a file named *p2q6.sql*. Rerun your query.

Employee	Monthly Salary
KING	5000
BLAKE	2850
CLARK	2450
ALLEN	1600

7. Display the name and hire date of every employee who was hired in 1982.

ENAME	HIREDATE
SCOTT	09-DEC-82
MILLER	23-JAN-82

8. Display the name and job title of all employees who do not have a manager.

ENAME	JOB
KING	PRESIDENT

9. Display the name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

ENAME	SAL	COMM
ALLEN	1600	300
TURNER	1500	0
MARTIN	1250	1400
WARD	1250	500

10. Display the names of all employees where the third letter of their name is an *A*.

ENAME
BLAKE
CLARK
ADAMS

11. Display the name of all employees who have two *L*s in their name and are in department 30 or their manager is 7782.

ENAME
ALLEN
MILLER

12. Display the name, job, and salary for all employees whose job is Clerk or Analyst and their salary is not equal to $1000, $3000, or $5000.

ENAME	JOB	SAL
JAMES	CLERK	950
SMITH	CLERK	800
ADAMS	CLERK	1100
MILLER	CLERK	1300

Lecture Notes (Lecture05_IDB)

- For details go through Database System Concepts, 6th Edition, Chapter 07

Lecture Notes (Lecture06_IDB)

Objective:

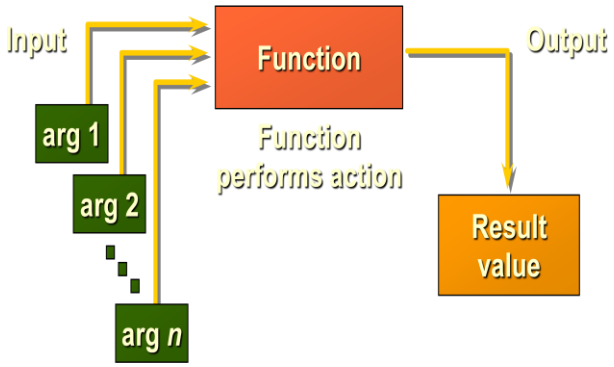
Functions make the basic query block more powerful and are used to manipulate data values. This is the first of two lessons that explore functions. You will focus on single-row character, number, and date functions, as well as those functions that convert data from one type to another—for example, character data to numeric.

SQL Functions

Functions are a very powerful feature of SQL and can be used to do the following:

- Perform calculations on data
- Modify individual data items
- Manipulate output for groups of rows
- Format dates and numbers for display
- Convert column datatypes
- SQL functions may accept arguments and always return a value.

Note: Most of the functions described in this lesson are specific to Oracle's version of SQL.



There are two distinct types of functions:

- Single-row functions
- Multiple-row functions

Single-Row Functions: These functions operate on single rows only and return one result per row. There are different types of single-row functions. This lesson covers the following ones:

- Character
- Number
- Date
- Conversion

Multiple-Row Functions: These functions manipulate groups of rows to give one result per group of rows.

Single-Row Functions

Single-row functions are used to manipulate data items. They accept one or more arguments and return one value for each row returned by the query. An argument can be one of the following:

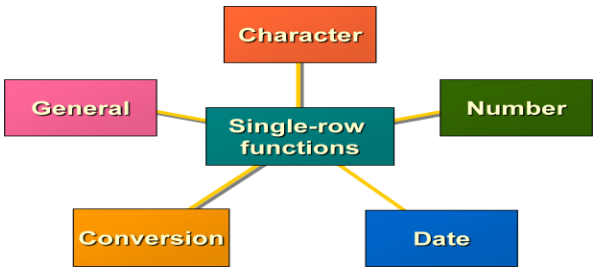
- User-supplied constant
- Variable value
- Column name
- Expression

Features of single-row functions:

- Act on each row returned in the query
- Return one result per row
- May return a data value of a different type than that referenced
- May expect one or more arguments
- Can be used in SELECT, WHERE, and ORDER BY clauses; can be nested

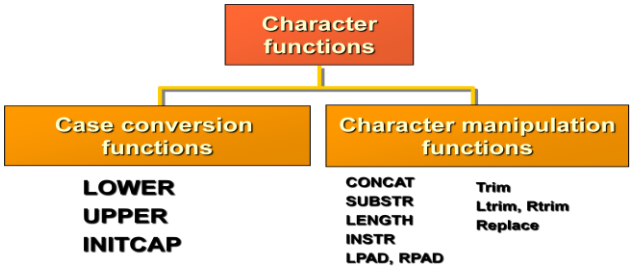
In the syntax:

function_name is the name of the function
column is any named database column
expression is any character string or calculated expression
arg1, arg2 is any argument to be used by the function



This lesson covers the following single-row functions:

- ***Character functions:*** Accept character input and can return both character and number values
- ***Number functions:*** Accept numeric input and return numeric values
- ***Date functions:*** Operate on values of the date datatype (All date functions return a value of date datatype except the MONTHS_BETWEEN function, which returns a number.)
- ***Conversion functions:*** Convert a value from one datatype to another
- ***General functions:***
  NVL function
  DECODE function



Convert case for character strings

Function	Result
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

LOWER, UPPER, and INITCAP are the three case conversion functions.

- LOWER: Converts mixed case or uppercase character string to lowercase
- UPPER: Converts mixed case or lowercase character string to uppercase
- INITCAP: Converts first letter of each word to uppercase and remaining letters to lowercase

The example displays the employee number, name, and department number of employee BLAKE. The WHERE clause of the first SQL statement specifies the employee name as 'blake.' Since all the data in the EMP table is stored in uppercase, the name 'blake' does not find a match in the EMP table and as a result no rows are selected. The WHERE clause of the second SQL statement specifies that the employee name in the EMP table be converted to lowercase and then be compared to 'blake.' Since both the names are in lowercase now, a match is found and one row is selected. The WHERE clause can be rewritten in the following manner to produce the same result:

```
SQL> SELECT   empno, ename, deptno
  2  FROM     emp
  3  WHERE    ename = 'blake';
no rows selected
```

```
SQL> SELECT   empno, ename, deptno
  2  FROM     emp
  3  WHERE    LOWER(ename) = 'blake';

     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      7698 BLAKE              30
```

The name in the output appears as it was stored in the database. To display the name with the first letter capitalized, use the INITCAP function in the SELECT statement.

Character Manipulation Functions

CONCAT, SUBSTR, LENGTH, INSTR, and LPAD are the five character manipulation functions covered in this lesson.

- ***CONCAT:*** Joins values together (You are limited to using two parameters with CONCAT.)
- ***SUBSTR:*** Extracts a string of determined length
- ***LENGTH:*** Shows the length of a string as a numeric value
- ***INSTR:*** Finds numeric position of a named character
- ***LPAD:*** Pads the character value right-justified

Note: RPAD character manipulation function pads the character value left-justified

Manipulate character strings

Function	Result
CONCAT(' Good ' , ' String ')	GoodString
SUBSTR(' String ' ,1,3)	Str
LENGTH(' String ')	6
INSTR(' String ' , ' r ')	3
LPAD(sal,10, ' * ')	******5000
Trim('S' from 'SSMITH')	MITH
Replace('toy','y','let')	

The example displays employee name and job joined together, length of the employee name, and the numeric position of the letter A in the employee name, for all employees who are in sales.

```
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

Example

Modify the SQL statement to display the data for those employees whose names end with an N.

Number Functions

Number functions accept numeric input and return numeric values. This section describes some of the number functions.

```
SQL> SELECT ROUND(45.923,2), ROUND(45.923,0),
  2         ROUND(45.923,-1)
  3  FROM   DUAL;
```

ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
45.92	46	50

ROUND Function

The ROUND function rounds the column, expression, or value to *n* decimal places. If the second argument is 0 or is missing, the value is rounded to zero decimal places. If the second argument is 2, the value is rounded to two decimal places. Conversely, if the second argument is -2, the value is rounded to two decimal places to the left. The ROUND function can also be used with date functions. You will see examples later in this lesson. The DUAL is a dummy table. More about this will be covered later.

TRUNC Function

The TRUNC function truncates the column, expression, or value to *n* decimal places. The TRUNC function works with arguments similar to those of the ROUND function. If the second argument is 0 or is missing, the value is truncated to zero decimal places. If the second argument is 2, the value is truncated to two decimal places. Conversely, if the second argument is -2, the value is truncated to two decimal places to the left. Like the ROUND function, the TRUNC function can be used with date functions.

```
SQL> SELECT TRUNC(45.923,2), TRUNC(45.923),
  2         TRUNC(45.923,-1)
  3  FROM   DUAL;
```

```
TRUNC(45.923,2) TRUNC(45.923) TRUNC(45.923,-1)
--------------- ------------- ---------------
          45.92            45              40
```

MOD Function

The MOD function finds the remainder of value1 divided by value2. The example calculates the remainder of the ratio of salary to commission for all employees whose job title is salesman.

```
SQL> SELECT   ename, sal, comm, MOD(sal, comm)
  2  FROM     emp
  3  WHERE    job = 'SALESMAN';
```

```
ENAME             SAL       COMM MOD(SAL,COMM)
---------- --------- --------- --------------
MARTIN           1250      1400           1250
ALLEN            1600       300            100
TURNER           1500         0           1500
WARD             1250       500            250
```

Working with Dates: *Oracle Date Format*

Oracle stores dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds. The default display and input format for any date is DD-MON-YY. Valid Oracle dates are between January 1, 4712 B.C., and December 31, 9999 A.D.

SYSDATE

SYSDATE is a date function that returns the current date and time. You can use SYSDATE just as you would use any other column name. For example, you can display the current date by selecting SYSDATE from a table. It is customary to select SYSDATE from a dummy table called DUAL.

DUAL

The DUAL table is owned by the user SYS and can be accessed by all users. It contains one column, DUMMY, and one row with the value X. The DUAL table is useful when you want to return a value once only—for instance, the value of a constant, pseudocolumn, or expression that is not derived from a table with user data.

Arithmetic with Dates

Since the database stores dates as numbers, you can perform calculations using arithmetic operators such as addition and subtraction. You can add and subtract number constants as well as dates. You can perform the following operations:

```
SQL> SELECT ename, (SYSDATE-hiredate)/7 WEEKS
  2  FROM   emp
  3  WHERE  deptno = 10;
```

```
ENAME          WEEKS
---------- ---------
KING       830.93709
CLARK      853.93709
MILLER     821.36566
```

Date Functions

Date functions operate on Oracle dates. All date functions return a value of DATE datatype except MONTHS_BETWEEN, which returns a numeric value.

- ***MONTHS_BETWEEN(date1, date2):*** Finds the number of months between *date1* and *date2*. The result can be positive or negative. If *date1* is later than *date2*, the result is positive; if *date1* is earlier than *date2*, the result is negative. The noninteger part of the result represents a portion of the month.
- ***ADD_MONTHS(date, n):*** Adds *n* number of calendar months to *date*. The value of *n* must be an integer and can be negative.
- ***NEXT_DAY(date, 'char'):*** Finds the date of the next specified day of the week ('*char*') following *date*. The value of *char* may be a number representing a day or a character string.
- ***LAST_DAY(date):*** Finds the date of the last day of the month that contains *date*.
- ***ROUND(date[,'fmt']):*** Returns *date* rounded to the unit specified by the format model *fmt.* If the format model *fmt* is omitted, *date* is rounded to the nearest day.
- ***TRUNC(date[, 'fmt']):*** Returns *date* with the time portion of the day truncated to the unit specified by the format model *fmt*. If the format model *fmt* is omitted, *date* is truncated to the nearest day.

For all employees employed for fewer than 200 months, display the employee number, hire date, number of months employed, six-month review date, first Friday after hire date, and last day of the month when hired.

```
SQL>  SELECT empno, hiredate,
      MONTHS_BETWEEN(SYSDATE, hiredate) TENURE,
      ADD_MONTHS(hiredate, 6) REVIEW,
      NEXT_DAY(hiredate, 'FRIDAY'), LAST_DAY(hiredate)
      FROM emp
      WHERE MONTHS_BETWEEN (SYSDATE, hiredate)<200;
```

The ROUND and TRUNC functions can be used for number and date values. When used with dates, these functions round or truncate to the specified format model. Therefore, you can round dates to the nearest year or month.

Example

Compare the hire dates for all employees who started in 1982. Display the employee number, hire date, and month started using the ROUND and TRUNC functions.

```
SQL>  SELECT empno, hiredate,
      ROUND(hiredate, 'MONTH'), TRUNC(hiredate, 'MONTH')
      FROM emp
      WHERE hiredate like '%82';
```

Conversion Functions

In addition to Oracle datatypes, columns of tables in an Oracle8 database can be defined using ANSI, DB2, and SQL/DS datatypes. However, the Oracle Server internally converts such datatypes to Oracle8 datatypes. In some cases, Oracle Server allows data of one datatype where it expects data of a different datatype. This is allowed when Oracle Server can automatically convert the data to the expected datatype. This datatype conversion can be done *implicitly* by Oracle Server or *explicitly* by the user. Implicit datatype conversions work according to the rules explained below. Explicit datatype conversions are done by using the conversion functions. Conversion functions convert a value from one datatype to another. Generally, the form of the function names follows the convention *datatype* TO *datatype*. The first datatype is the input datatype; the last datatype is the output.

Note: Although implicit datatype conversion is available, it is recommended that you do explicit datatype conversion to ensure reliability of your SQL statements.

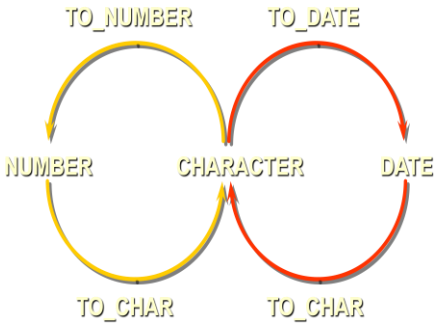
Implicit Datatype Conversion

In general, the Oracle Server uses the rule for expression when a datatype conversion is needed in places not covered by a rule for assignment conversions.

Note: CHAR to NUMBER conversions succeed only if the character string represents a valid number. CHAR to DATE conversions succeed only if the character string has the default format DD-MON-YY.

Explicit Datatype Conversion

SQL provides three functions to convert a value from one datatype to another.



Displaying a Date in a Specific Format

Previously, all Oracle date values were displayed in the DD-MON-YY format. The TO_CHAR function allows you to convert a date from this default format to one specified by you.

Guidelines

- The format model must be enclosed in single quotation marks and is case sensitive.
- The format model can include any valid date format element. Be sure to separate the date value from the format model by a comma.
- The names of days and months in the output are automatically padded with blanks.
- To remove padded blanks or to suppress leading zeros, use the fill mode *fm* element.
- You can resize the display width of the resulting character field with the SQL*Plus COLUMN command.
- The resultant column width is 80 characters by default.

YYYY	**Full year in numbers**
YEAR	**Year spelled out**
MM	**Two-digit value for month**
MONTH	**Full name of the month**
DY	**Three-letter abbreviation of the day of the week**
DAY	**Full name of the day**

TO_CHAR(value, [format_mask], [nls_language])

Value can either be a number or date that will be converted to a string. Format mask is optional. This is the format that will be used to convert value to a string. nls_language is optional. This is the nls language used to convert value to a string.

TO_CHAR(sysdate, 'yyyy/mm/dd'); would return '2003/07/09'
TO_CHAR(sysdate, 'Month DD, YYYY'); would return 'July 09, 2003'
TO_CHAR(sysdate, 'FMMonth DD, YYYY'); would return 'July 9, 2003'
TO_CHAR(sysdate, 'MON DDth, YYYY'); would return 'JUL 09TH, 2003'
TO_CHAR(sysdate, 'FMMON DDth, YYYY'); would return 'JUL 9TH, 2003'
TO_CHAR(sysdate, 'FMMon ddth, YYYY'); would return 'Jul 9th, 2003'

TO_CHAR Function with Numbers

When working with number values such as character strings, you should convert those numbers to the character datatype using the TO_CHAR function, which translates a value of NUMBER datatype to VARCHAR2 datatype. This technique is especially useful with concatenation.

TO_CHAR(number, 'fmt')

9	Represents a number
0	Forces a zero to be displayed
$	Places a floating dollar sign
L	Uses the floating local currency symbol
.	Prints a decimal point
,	Prints a thousand indicator

```
SQL> SELECT  TO_CHAR(sal,'$99,999') SALARY
  2  FROM    emp
  3  WHERE   ename = 'SCOTT';
```

```
SALARY
--------
  $3,000
```

TO_DATE (string1, [format_mask], [nls_language])

string1 is the string that will be converted to a date. *Format mask* is optional. This is the format that will be used to convert *string1* to a date. *nls_language* is optional. This is the nls language used to convert *string1* to a date.

TO_DATE('2003/07/09', 'yyyy/mm/dd') would return a date value of July 9, 2003
TO_DATE('070903', 'MMDDYY') would return a date value of July 9, 2003
TO_DATE('20020315', 'yyyymmdd') would return a date value of Mar 15, 2002

TO_NUMBER and TO_DATE Functions

You may want to convert a character string to either a number or a date. To accomplish this task, you use the TO_NUMBER or TO_DATE functions. The format model you choose will be based on the previously demonstrated format elements.

```
TO_NUMBER(char[, 'fmt'])
```

```
TO_DATE(char[, 'fmt'])
```

Example

Display the names and hire dates of all the employees who joined on February 22, 1981.

The NVL Function

To convert a null value to an actual value, use the NVL function.

Syntax

NVL (*expr1*, *expr2*)

Where: *expr1* is the source value or expression that may contain null

expr2 is the target value for converting null

You can use the NVL function to convert any datatype, but the return value is always the same as the datatype of *expr1*.

```
SQL> SELECT ename, sal, comm, (sal*12)+NVL(comm,0)
  2  FROM   emp;
```

```
ENAME             SAL       COMM (SAL*12)+NVL(COMM,0)
---------- --------- --------- --------------------
KING             5000                         60000
BLAKE            2850                         34200
CLARK            2450                         29400
JONES            2975                         35700
MARTIN           1250       1400              16400
ALLEN            1600        300              19500
...
14 rows selected.
```

NVL2 (expr1,expr2,expr3)

If exp1 is not null it returns exp2, If exp1 is null it returns exp3

Example:

Select nvl2(comm,comm+sal,sal) from emp;

NULLIF(expr1,expr2)

Compares 2 expressions and returns null if they are equal or the first expression if they are not equal.

Example:

Select ename, length(ename), job, length(job), NULLIF(ename,job) from emp;

The DECODE Function

The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic used in various languages. The DECODE function decodes *expression* after comparing it to each *search* value. If the expression is the same as *search*, *result* is returned. If the default value is omitted, a null value is returned where a search value does not match any of the result values.

```
SQL> SELECT job, sal,
  2         DECODE(job, 'ANALYST',  SAL*1.1,
  3                     'CLERK',    SAL*1.15,
  4                     'MANAGER',  SAL*1.20,
  5                                 SAL)
  6                REVISED_SALARY
  7  FROM   emp;
```

```
JOB             SAL REVISED_SALARY
--------- --------- --------------
PRESIDENT       5000           5000
MANAGER         2850           3420
MANAGER         2450           2940
...
14 rows selected.
```

In the SQL statement above, the value of JOB is decoded. If JOB is ANALYST, the salary increase is 10%; if JOB is CLERK, the salary increase is 15%; if JOB is MANAGER, the salary increase is 20%. For all other job roles, there is no increase in salary.

Nesting Functions

Single-row functions can be nested to any depth. Nested functions are evaluated from the innermost level to the outermost level. Some examples follow to show you the flexibility of these functions.

```
SQL> SELECT   ename,
  2           NVL(TO_CHAR(mgr),'No Manager')
  3  FROM     emp
  4  WHERE    mgr IS NULL;
```

```
ENAME      NVL(TO_CHAR(MGR),'NOMANAGER')
---------- ----------------------------
KING       No Manager
```

Exercise:

1. Write a query to display the current date. Label the column Date.
2. Display the employee number, name, salary, and salary increase by 15% expressed as a whole number. Label the column New Salary.
3. Modify your previous query to add a column that will subtract the old salary from the new salary. Label the column Increase. Rerun your query.
4. Display the employee's name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear in the format similar to "Sunday, the Seventh of September, 1981."
5. For each employee display the employee name and calculate the number of months between today and the date the employee was hired. Label the column MONTHS_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.
6. Write a query that produces the following for each employee: <employee name> earns <salary> monthly but wants <3 times salary>. Label the column Dream Salaries.
7. Write a query that will display the employee's name with the first letter capitalized and all other letters lowercase and the length of their name, for all employees whose name starts with J, A, or M. Give each column an appropriate label.
8. Create a query that will display the employee name and commission amount. If the employee does not earn commission, put "No Commission." Label the column COMM.
9. Create a query that displays the employees' names and indicates the amounts of their salaries through asterisks. Each asterisk signifies a hundred dollars. Sort the data in descending order of salary. Label the column EMPLOYEE_AND_THEIR_SALARIES.

Lecture Notes (Lecture07_IDB)

1. Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.
2. Design an E-R diagram for keeping track of the exploits of your favorite sports team. You should store the matches played, the scores in each match, the players in each match and individual player statistics for each match. Summary statistics should be modeled as derived attributes. 5. Extend the E-R diagram of the previous question to track the same information for all teams in a league.
3. A large bank operates several divisions. Information Technology (IT) is operated as one of these divisions. Within the IT division are many departments that are managed by one manager, and all IT employees belong to one of these departments. The IT division assigns its employees to one or more ongoing projects in the bank. A project may be planned, but not have any employees assigned to it for several months. Each project will have a single employee assigned who acts as a project leader.
4. A hardware store sells several home workshop products to the public (such as power saws and sanders). Each product has several different manufacturers who manufacture it, and prices are different for products made by different manufacturers. Each time one or more products are sold to a customer, an invoice is created which lists the date, items purchased and their prices, and then the total purchase and tax amounts.
5. The Ministry of Transportation (MOT) supplies department keeps track of all the items (furniture and equipment such as a chair or printer) in the Ministry offices. There are several MOT buildings and each one is given a different name to identify it. Each item is assigned a unique ID when it is purchased. This ID is used to keep track of the item, which is assigned to a room within a building. Each room within a building is assigned to a department, and each department has a single employee as its manager.
6. A cooking club organizes several dinners for its members. The purpose of the club is to allow several members to get together and prepare a dinner for the other members. The club president maintains a database that plans each meal and tracks which members attends each dinner, and also keeps track of which members creates each dinner. Each dinner serves many members and any member is allowed to attend. Each dinner has an invitation. This invitation is mailed to each member. The invitation includes the date of the dinner and location. Each dinner is based on a single entrée and a single dessert. This entrée and dessert can be used again for other dinners.
7. ABC Consulting is a small-sized consulting firm in the IT industry. ABC's business is managing several Systems Development projects by assigning staff consultants to these projects as their skills are needed. Each employee is designated to have one primary skill, but there may be other employees with the same primary skill. A consultant may work on one or more projects, or may not yet be assigned to a project. The company charges for each project by billing each consultant's hours worked by the billing rate. The hourly billing rate is dependent on the employee's primary job skill.
8. A company purchases products and sells them to its customers. Each time a sale occurs, an invoice is created listing the customer name, and a list of purchase product descriptions, the supplier name for the products, and the price of each product. The product number identifies each product and will

appear again if another customer purchases the same product. Each supplier can supply many products which we can sell, but each product has only one supplier.

9. You are asked to create a database to produce a report of customer details. The report is to list the customer name, account balance, credit limit, and other customer details. The report will also list the customer account rep (one of our sales employees). Our sales reps manage many customers each, but each customer will be managed by only one account rep at any one time. However, your design should allow for customer's being managed by many account reps as it is possible that some of our employees may leave the company - thus, requiring new account reps for a customer.

10. A company operates a warehouse parts supply business. The company has several warehouses located in Toronto which each store several hundreds of automotive parts. We need to keep a record of how many parts are "on hand" - meaning inventory levels that tell us how many we have for each part. To help us organize our parts, each part is assigned a specific classification. There are 4 classifications that we use to organize hundreds of parts.

11. To keep track of office furniture, computers, printers, and so on, the FOUNDIT Company requires the creation of a simple database. Each piece of office furniture, computer or printer is given an identification number. Each item is then placed in a room of one of three buildings. The building manager is responsible for the items in their building.

Lecture Notes (Lecture08_IDB)

Aggregate functions return a single result row based on groups of rows, rather than on single rows. Aggregate functions can appear in select lists and in ORDER BY and HAVING clauses. They are commonly used with the GROUP BY clause in a SELECT statement, where Oracle Database divides the rows of a queried table or view into groups. In a query containing a GROUP BY clause, the elements of the select list can be aggregate functions, GROUP BY expressions, constants, or expressions involving one of these. Oracle applies the aggregate functions to each group of rows and returns a single result row for each group.

Types of Group Functions:

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE

SUM function: used to return the sum of an expression in a SELECT statement.

Syntax: *SELECT SUM(<col>) FROM <table> WHERE conditions;*

AVG Function: used to return the average of an expression in a SELECT statement.

Syntax: *SELECT AVG(<col>) FROM <table> WHERE conditions;*

COUNT Function: used to count the number of rows returned in a SELECT statement.

Syntax: *SELECT COUNT(<col>) FROM <table> WHERE conditions;*

MAX Function: used to return the maximum value of an expression in a SELECT statement.

Syntax: *SELECT MAX(<col>) FROM <table> WHERE conditions;*

MIN Function: used to return the minimum value of an expression in a SELECT statement.

Syntax: SELECT MIN(<col>) FROM <table> WHERE conditions;

*** <col> can be a numeric field or a formula

The SQL **GROUP BY** clause can be used in a SELECT statement to collect data across multiple records and group the results by one or more columns.

Syntax: *SELECT <aggregate function>, <col> FROM <table> GROUP BY <col>;*

The **HAVING clause** was added to SQL because the WHERE keyword could not be used with aggregate functions.

Syntax: *SELECT <aggregate function>, <col> FROM <table> GROUP BY <col> HAVING <condition>;*

Exercise

Reference Table: Oracle built in tables.

1. Find average, maximum, minimum salary of the employees.
2. Find average, maximum, minimum salary of the employees according to department number.
3. Find average, maximum, minimum salary of the employees according to job category.
4. Find the name of lowest paid manager. (Manager is not Job).
5. Find the location where maximum number of employee is located
6. Find out job group having highest amount of total salary. (Sal + comm)
7. Suppose you need to know the name and department no. of the employee who earns the highest salary. Write a SQL query to return this information.

** Please save the SQL commands in a text file for further use.

Lecture Notes (Lecture09_IDB)

Objectives:

To learn Data Definition Language (DDL) commands which allow you to perform these tasks:

- create, alter, and drop objects
- grant and revoke privileges and roles

Data Types

SQL is structure query language.SQL contains different data types those are:

- char(size)
- varchar(size)
- varchar2(size)
- date
- number(p,s) //** p-PRECISION s-SCALE **//
- number(size)
- raw(size)
- raw/long raw(size)

Create Table

It defines each column of the table uniquely. Each column has minimum of three attributes, a **name** , **data type** and **size**.

Syntax: Create table <table name> (<col1> <datatype>(<size>),<col2><datatype><size>));

Modifying the structure of tables

Add new columns:

Syntax: Alter table <tablename> add(<new col> <datatype(size), <newcol> datatype(size));

Dropping a column from a table

Syntax: Alter table <tablename> drop column <col>;

Modifying existing columns

Syntax: Alter table <tablename> modify(<col><newdatatype>(<newsize>));

Renaming the tables

Syntax: Rename <oldtable> to <new table>;

Truncating the tables

Syntax: Truncate table <tablename>;

Destroying tables

Syntax: Drop table <tablename>;

Data Manipulation Language (DML) statements are used for managing data within schema objects.

- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain
- MERGE - UPSERT operation (insert or update)

Inserting Data into Tables: Once a table is created the most natural thing to do is load this table with data to be manipulated later.

Syntax 1: insert into <tablename> (<col1>,<col2>…..<col n>) values(<val 1>,<val 2>…….<val n>); Syntax 2(takes input from user): insert into <tablename> values(&<col1>,&<col2>……,&<col n>);

Syntax 3: insert into <tablename> values(<val 1>,<val 2>…….,<val n>);

Delete operations

a) Remove all rows:

Syntax: delete from <tablename>;

b) Removal of a specified row/s:

Syntax: delete from <tablename> where <condition>;

Updating the contents of a table

a) updating all rows:

Syntax: Update <tablename> set <col>=<exp>,<col>=<exp>;

b) Updating selected records:

Syntax: Update <tablename> set <col>=<exp>,<col>=<exp> where <condition>;

Exercise

1. Create a table named **Student** from following structure using SQL commands:

Column Name	Data Type
s_id	Number
s_name	Varchar2(20)
phone	number
address	Varchar2(50)
email	Varchar2(30)
credit_completed	Number(3)
course_completed	Number(2)
cgpa	Number

2. Add following columns into the above **student** table:

Column Name	Data Type
department	Varchar2(5)
gender	Varchar2(6)

3. Modify the column name **department** into **dept**.
4. Change type of the column **cgpa** into **number(2,3).**
5. Drop column **email** from student table.
6. Change the table name from **student** to **students**.

Reference Table:

Department

Column name	Data type	Constraint
deptid	number(3)	primary key
dept_name	varchar(6)	only CSE, EEE, BBA, Eng, Ach allowed
budget	number(6)	default value 0

Course

Column name	Data type	Constraint
crs_id	number(4)	primary key
crs_name	varchar2(20)	not null
dept_id	number(3)	foreign key from department table

7. Insert 4 rows into **department** table.
8. Insert 3 rows into **course** table.
9. Insert a row into department so it take dept_name and budget as input from user.
10. Update budget of 'CSE' department to 4000$.
11. Delete all courses from course table.

Lecture Notes (Lecture10_IDB)

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database. Constraints could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL:

- ***NOT NULL*** Constraint: Ensures that a column cannot have NULL value.
- ***DEFAULT*** Constraint: Provides a default value for a column when none is specified.
- ***UNIQUE*** Constraint: Ensures that all values in a column are different.
- ***PRIMARY Key***: Uniquely identified each rows/records in a database table.
- ***FOREIGN Key***: Uniquely identified a rows/records in any another database table.
- ***CHECK*** Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.

Constraints can be specified when a table is created with the **CREATE TABLE** statement or you can use **ALTER TABLE** statement to create constraints even after the table is created.

NOT NULL constraint:

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.
Syntax: <col><datatype>(size)not null

UNIQUE constraint:

The UNIQUE constraint uniquely identifies each record in a database table.
Syntax: <col><datatype>(size)unique;

PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain unique values. A primary key column cannot contain NULL values. Each table should have a primary key, and each table can have only ONE primary key.

Syntax: <col><datatype>(size)primary key
Or,
constraint <constraint_name> primary key(<col1>, <col2>)

FOREIGN KEY Constraint

FOREIGN KEY in one table points to a PRIMARY KEY in another table.
Syntax: <col><datatype>(<size>) foreign key references <tablename>(<col>)
Or,
constraint <constraint_name> foreign key<current table col> references <reference table name>(<reference col>)

CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column.
Syntax: <col><datatype>(size) check(<logical expression>)
Or,
constraint <constraint_name> check <col> (<logical expression>)

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

In previous lab we have learnt to add constraints on the time of create the table. You can also add or change constraint after table creates.

To add a constraint:

Syntax: alter table <table>

add constraint <constraint_name> <constraint type> (<col>);

**for foreign key:

alter table <table>

add <constraint_name> foreign key<current table col> references

<reference table name>(<reference col>);

To drop a constraint:

Syntax: alter table <table>

drop constraint <constraint_name>;

To disable a constraint:

Syntax: alter table <table>

disable constraint <constraint_name>;

To enable a constraint:

Syntax: alter table <table>

enable constraint <constraint_name>;

To viewing the Columns Associated with Constraints:

Syntax: select constraint_name, column_name from user_cons_columns where

table_name = '<table>';

Exercise

1. Create following **department** table according to given data type and constraints:

Column name	Data type	Constraint
deptid	number(3)	primary key
dept_name	varchar(6)	only CSE, EEE, BBA, Eng, Ach allowed
budget	number(6)	default value 0

2. Create following **course** table according to givent data type and constraint:

Column name	Data type	Constraint
crs_id	number(4)	primary key
crs_name	varchar2(20)	not null
dept_id	number(3)	foreign key from department table

3.

Column Name	Data Type
s_id	Number
s_name	Varchar2(20)
phone	number
address	Varchar2(50)
email	Varchar2(30)
credit_completed	Number(3)
course_completed	Number(2)
cgpa	Number
deptno	number(5)
gender	Varchar2(6)

Create above table according to given data types.

4. Set **s_id** as primary key of the table.
5. Set constraint not null on the column **s_name.**
6. Make **email** unique.
7. Make **deptno** as foreign key taking reference from **department** table which you have made in previous lab.
8. Add a constraint to **gender** so that it only allows the value **'M'** and **'F'**.

9. Disable the constraint of **s_id**.
10. Drop the constraint from **gender**.
11. View the columns associated with constraints.
12. Enable the constraint of **s_id**.

** Please save the SQL commands in a text file for further use.

Lecture Notes (Lecture11_IDB)

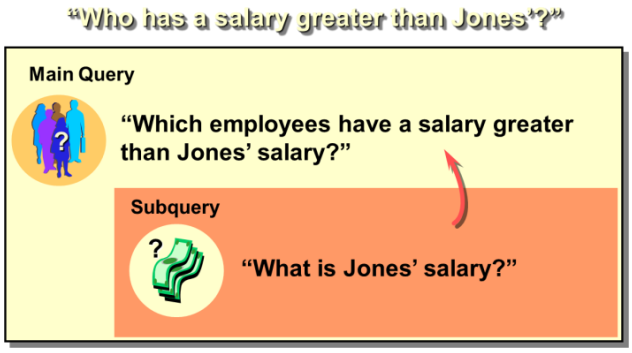
Objective:

- Describe the types of problems that subqueries can solve
- Define subqueries
- List the types of subqueries
- Write single-row subqueries

In this lesson, you will learn about more advanced features of the SELECT statement. You can write subqueries in the WHERE clause of another SQL statement to obtain values based on an unknown conditional value. This lesson covers single-row subqueries and multiple-row subqueries.

Using a Subquery to Solve a Problem

Suppose you want to write a query to find out who earns a salary greater than Jones' salary. To solve this problem, you need *two* queries: one query to find what Jones earns and a second query to find who earns more than that amount. You can solve this problem by combining the two queries, placing one query *inside* the other query. The inner query or the *subquery* returns a value that is used by the outer query or the main query. Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the search value in the second query.



Subqueries

A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. You can build powerful statements out of simple ones by using subqueries. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.
You can place the subquery in a number of SQL clauses:

- WHERE clause
- HAVING clause
- FROM clause

In the syntax:

operator includes a comparison operator such as >, =, or IN

Note: Comparison operators fall into two classes: single-row operators (>, =, >=, <, <>, <=) and multiple-row operators (IN, ANY, ALL). The subquery is often referred to as a nested SELECT, sub-SELECT, or inner SELECT statement. The subquery generally executes first, and its output is used to complete the query condition for the main or outer query.

```
SQL> SELECT ename
  2  FROM   emp
  3  WHERE  sal > 
  4                (SELECT sal
  5                 FROM   emp
  6                 WHERE  empno=7566);
```

2975

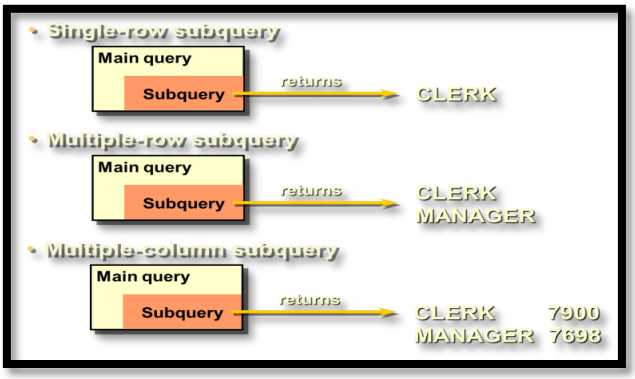
```
ENAME
----------
KING
FORD
SCOTT
```

Guidelines for Using Subqueries

- A subquery must be enclosed in parentheses.
- A subquery must appear on the right side of the comparison operator.
- Subqueries cannot contain an ORDER BY clause. You can have only one ORDER BY clause for a SELECT statement, and if specified it must be the last clause in the main SELECT statement.
- Two classes of comparison operators are used in subqueries: single-row operators and multiple-row operators.

Types of Subqueries

- Single-row subqueries: Queries that return only one row from the inner SELECT statement
- Multiple-row subqueries: Queries that return more than one row from the inner SELECT statement
- Multiple-column subqueries: Queries that return more than one column from the inner SELECT statement



Single-Row Subqueries

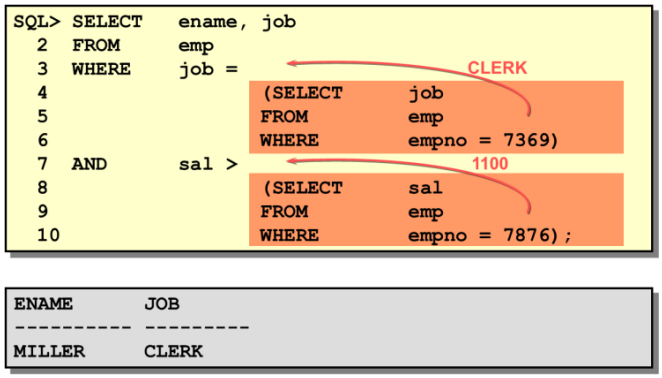
A *single-row subquery* is one that returns one row from the inner SELECT statement. This type of subquery uses a single-row operator.

Example
Display the employees whose job title is the same as that of employee 7369.

Executing Single-Row Subqueries
A SELECT statement can be considered as a query block. The example on the displays employees whose job title is the same as that of employee 7369 and whose salary is greater than that of employee 7876.

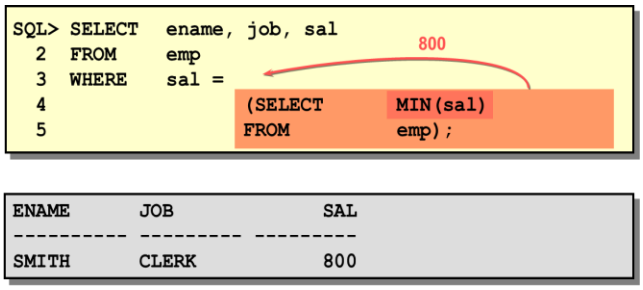
The example consists of three query blocks: the outer query and two inner queries. The inner query blocks are executed first, producing the query results: CLERK and 1100, respectively. The outer query block is then processed and uses the values returned by the inner queries to complete its search conditions.

Both inner queries return single values (CLERK and 1100, respectively), so this SQL statement is called a single-row subquery.



Using Group Functions in a Subquery
You can display data from a main query by using a group function in a subquery to return a single row. The subquery is in parentheses and is placed after the comparison operator.

The example displays the employee name, job title, and salary of all employees whose salary is equal to the minimum salary. The MIN group function returns a single value (800) to the outer query.



HAVING Clause with Subqueries

You can use subqueries not only in the WHERE clause, but also in the HAVING clause. The Oracle Server executes the subquery, and the results are returned into the HAVING clause of the main query. The SQL statement stated below displays all the departments that have a minimum salary greater than that of department 20.

```
SQL> SELECT       deptno, MIN(sal)
  2  FROM         emp
  3  GROUP BY     deptno
  4  HAVING       MIN(sal) >
  5                       (SELECT    MIN(sal)
  6                       FROM       emp
  7                       WHERE      deptno = 20);
```

800

Errors with Subqueries

One common error with subqueries is more than one row returned for a single-row subquery. In the SQL statement shown below, the subquery contains a GROUP BY (deptno) clause, which implies that the subquery will return multiple rows, one for each group it finds. In this case, the result of the subquery will be 800, 1300, and 950. The outer query takes the results of the subquery (800, 950, 1300) and uses these results in its WHERE clause. The WHERE clause contains an equal (=) operator, a single-row comparison operator expecting only one value. The = operator cannot accept more than one value from the subquery and hence generates the error. To correct this error, change the = operator to IN.

```
SQL> SELECT empno, ename
  2  FROM   emp
  3  WHERE  sal =
  4                (SELECT    MIN(sal)
  5                FROM       emp
  6                GROUP BY   deptno);
```

Single-row operator with multiple-row subquery

```
ERROR:
ORA-01427: single-row subquery returns more than
one row

no rows selected
```

Problems with Subqueries

A common problem with subqueries is no rows being returned by the inner query. In the SQL statement given below, the subquery contains a WHERE (ename='SMYTHE') clause. Presumably, the intention is to find the employee whose name is Smythe. The statement seems to be correct but selects no rows when executed.

```
SQL> SELECT ename, job
  2  FROM   emp
  3  WHERE  job =
  4                (SELECT job
  5                FROM   emp
  6                WHERE  ename='SMYTHE');

no rows selected
```

The problem is that Smythe is misspelled. There is no employee named Smythe. So the subquery returns no rows. The outer query takes the results of the subquery (null) and uses these results in its WHERE clause. The outer query finds no employee with a job title equal to null and so returns no rows.

Exercise:

1. Display all the employees who are earning more than all the managers.
2. Display all the employees who are earning more than any of the managers.
3. Select employee number, job & salaries of all the Analysts who are earning more than any of the managers.
4. Select all the employees who work in DALLAS.
5. Select department name & location of all the employees working for CLARK.
6. Select all the departmental information for all the managers.