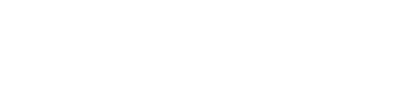
**RAJALAKSHMI ENGINEERING COLLEGE RAJALAKSHMI NAGAR, THANDALAM – 602 105**

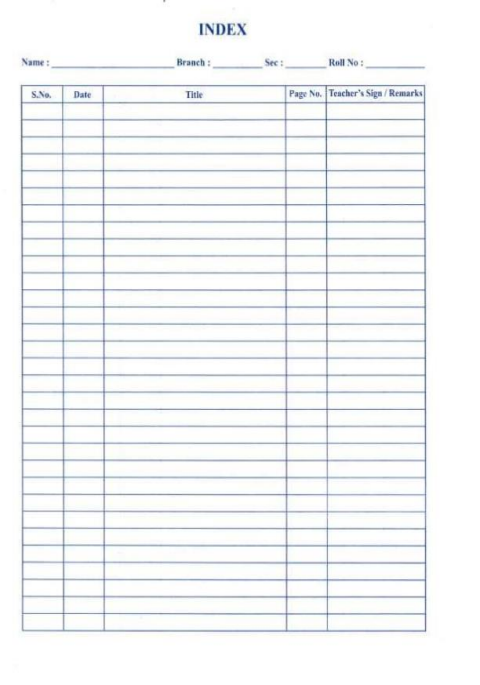
****

**CS23332 DATABASE MANAGEMENT SYSTEMS LAB** 

**Laboratory Record Note Book**

Name :…………………………. ……………………………... Year / Branch / Section : ……….. ………………………………. University Register No. : …………….. …………………………………… College Roll No. : …………………….…………………………………… Semester : ………………………………….……………………………………… Academic Year : ………………………………………………………….





**Definition of a Relational Database**

A relational database is a collection of relations or two-dimensional tables. **Terminologies Used in a Relational Database**

1. A single ROW or table representing all data required for a particular employee. Each row should be identified by a primary key which allows no duplicate rows.

2. A COLUMN or attribute containing the employee number which identifies a unique employee. Here Employee number is designated as a primary key ,must contain a value and must be unique.

3. A column may contain foreign key. Here Dept\_ID is a foreign key in employee table and it is a primary key in Department table.

4. A Field can be found at the intersection of a row and column. There can be only one value in it. Also it may have no value. This is called a null value.

| EMP ID | FIRST NAME | LAST NAME | EMAIL |
| --- | --- | --- | --- |
| 100 | King | Steven | Sking |
| 101 | John | Smith | Jsmith |
| 102 | Neena | Bai | Neenba |
| 103 | Eex | De Haan | Ldehaan |

**Relational Database Properties**

**A relational database :**

● Can be accessed and modified by executing structured query language (SQL) statements. ● Contains a collection of tables with no physical pointers.

● Uses a set of operators

**Relational Database Management Systems**

RDBMS refers to a relational database plus supporting software for managing users and processing SQL queries, performing backups/restores and associated tasks. (Relational Database Management System) Software for storing data using SQL (structured query language). A relational database uses SQL to store data in a series of tables that not only record existing relationships between data items, but which also permit the data to be joined in new relationships. SQL (pronounced 'sequel') is based on a system of algebra developed by E F Codd, an IBM scientist who first defined the relational model in 1970. Relational databases are optimized for storing transactional data, and the majority of modern business software applications therefore use an RDBMS as their data store. The leading RDBMS vendors are Oracle, IBM and Microsoft.

The first commercial RDBMS was the Multics Relational Data Store, first sold in 1978. INGRES, Oracle, Sybase, Inc., Microsoft Access, and Microsoft SQL Server are well known database products and companies.Others include PostgreSQL, SQL/DS, and RDB. A relational database management system (RDBMS) is a program that lets you create, update, and administer a relational database. Most commercial RDBMS's use the Structured Query Language (SQL) to access the database, although SQL was invented after the development of the relational model and is not necessary for its use.

The leading RDBMS products are Oracle, IBM's DB2 and Microsoft's SQL Server. Despite repeated challenges by competing technologies, as well as the claim by some experts that no current RDBMS has fully implemented relational principles, the majority of new corporate databases are still being created and managed with an RDBMS.

**SQL Statements**

1. Data Retrieval(DR)

2. Data Manipulation Language(DML)

3. Data Definition Language(DDL)

4. Data Control Language(DCL)

5. Transaction Control Language(TCL)

| **TYPE** | **STATEMENT** | **DESCRIPTION** |
| --- | --- | --- |
| DR | SELECT | Retrieves the data from the database |
| DML | 1.INSERT  2.UPDATE  3.DELETE  4.MERGE | Enter new rows, changes existing rows, removes unwanted rows from tables in the database respectively. |
| DDL | 1.CREATE  2.ALTER  3.DROP  4.RENAME  5.TRUNCATE | Sets up, changes and removes data structures from tables. |
| TCL | 1.COMMIT  2.ROLLBACK 3.SAVEPOINT | Manages the changes made by DML statements. Changes to the data can be grouped together into logical  transactions. |
| DCL | 1.GRANT  2.RREVOKE | Gives or removes access rights to both the oracle database and the structures within it. |

**DATA TYPES**

**1. Character Data types:**

▪ Char – fixed length character string that can varies between 1-2000 bytes ▪ Varchar / Varchar2 – variable length character string, size ranges from 1-4000 bytes.it saves the disk space(only length of the entered value will be assigned as the size of column) ▪ Long - variable length character string, maximum size is 2 GB

**2. Number Data types :** Can store +ve,-ve,zero,fixed point, floating point with 38 precission. ▪ Number – {p=38,s=0}

▪ Number(p) - fixed point

▪ Number(p,s) –floating point (p=1 to 38,s= -84 to 127)

**3. Date Time Data type:** used to store date and time in the table.

▪ DB uses its own format of storing in fixed length of 7 bytes for century, date, month, year, hour, minutes, and seconds.

▪ Default data type is “dd-mon-yy”

▪ New Date time data types have been introduced. They are

TIMESTAMP-Date with fractional seconds

INTERVAL YEAR TO MONTH-stored as an interval of years and months INTERVAL DAY TO SECOND-stored as o interval of days to hour’s minutes and seconds **4. Raw Data type:** used to store byte oriented data like binary data and byte string.

**5. Other :**

▪ CLOB – stores character object with single byte character.

▪ BLOB – stores large binary objects such as graphics, video, sounds. ▪ BFILE – stores file pointers to the LOB’s.

**EXERCISE-1**

**Creating and Managing Tables**

**OBJECTIVE**

After the completion of this exercise, students should be able to do the following: ⮚ Create tables

⮚ Describing the data types that can be used when specifying column definition ⮚ Alter table definitions

⮚ Drop, rename, and truncate tables

**NAMING RULES**

Table names and column names:

● Must begin with a letter

● Must be 1-30 characters long

● Must contain only A-Z, a-z, 0-9, \_, $, and #

● Must not duplicate the name of another object owned by the same user ● Must not be an oracle server reserve words

● 2 different tables should not have same name.

● Should specify a unique column name.

● Should specify proper data type along with width

● Can include “not null” condition when needed. By default it is ‘null’. **The CREATE TABLE Statement**

**Table:** Basic unit of storage; composed of rows and columns

**Syntax: 1** Create table table\_name (column\_name1 data\_ type (size)

column\_name2 data\_ type (size)….);

**Syntax: 2** Create table table\_name (column\_name1 data\_ type (size) constraints, column\_name2 data\_ type constraints …);

**Example:**

Create table employlees ( employee\_id number(6), first\_name varchar2(20), ..job\_id varchar2(10), CONSTRAINT emp\_emp\_id\_pk PRIMARY KEY (employlee\_id));

**Tables Used in this course**

**Creating a table by using a Sub query**

**SYNTAX**

// CREATE TABLE table\_name(column\_name type(size)…);

Create table table\_name **as** select column\_name1,column\_name2,……colmn\_namen from table\_name where predicate;

**AS Subquery**

Subquery is the select statement that defines the set of rows to be inserted into the new table.

**Example**

Create table dept80 as select employee\_id, last\_name, salary\*12 Annsal, hire\_date from employees where dept\_id=80;

**The ALTER TABLE Statement**

The ALTER statement is used to

● Add a new column

● Modify an existing column

● Define a default value to the new column

● Drop a column

● To include or drop integrity constraint.

**SYNTAX**

ALTER TABLE table\_name ADD /MODIFY(Column\_name type(size)); ALTER TABLE table\_name DROP COLUMN (Column\_nname);

*ALTER TABLE ADD CONSTRAINT Constraint\_name PRIMARY KEY (Colum\_Name);* **Example:**

Alter table dept80 add (jod\_id varchar2(9));

Alter table dept80 modify (last\_name varchar2(30));

Alter table dept80 drop column job\_id;

**NOTE:** Once the column is dropped it cannot be recovered.

**DROPPING A TABLE**

● All data and structure in the table is deleted.

● Any pending transactions are committed.

● All indexes are dropped.

● Cannot roll back the drop table statement.

**Syntax:**

**Drop table *tablename;***

**Example:**

Drop table dept80;

**RENAMING A TABLE**

To rename a table or view.

**Syntax**

RENAME old\_name to new\_name

**Example:**

Rename dept to detail\_dept;

**TRUNCATING A TABLE**

Removes all rows from the table.

Releases the storage space used by that table.

**Syntax**

TRUNCATE TABLE *table\_name*;

E**xample:**

TRUNCATE TABLE copy\_emp;

**Find the Solution for the following:**

**Create the following tables with the given structure.**

**EMPLOYEES TABLE**

| **NAME** | **NULL?** | **TYPE** |
| --- | --- | --- |
| Employee\_id | Not null | Number(6) |
| First\_Name |  | Varchar(20) |
| Last\_Name | Not null | Varchar(25) |
| Email | Not null | Varchar(25) |
| Phone\_Number |  | Varchar(20) |
| Hire\_date | Not null | Date |
| Job\_id | Not null | Varchar(10) |
| Salary |  | Number(8,2) |
| Commission\_pct |  | Number(2,2) |
| Manager\_id |  | Number(6) |
| Department\_id |  | Number(4) |

**DEPARTMENT TABLE**

| **NAME** | **NULL?** | **TYPE** |
| --- | --- | --- |
| Dept\_id | Not null | Number(6) |
| Dept\_name | Not null | Varchar(20) |
| Manager\_id |  | Number(6) |
| Location\_id |  | Number(4) |

**JOB\_GRADE TABLE**

| **NAME** | **NULL?** | **TYPE** |
| --- | --- | --- |
| Grade\_level |  | Varchar(2) |
| Lowest\_sal |  | Number |

| Highest\_sal |  | Number |
| --- | --- | --- |

**LOCATION TABLE**

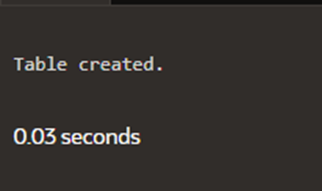
| **NAME** | **NULL?** | **TYPE** |
| --- | --- | --- |
| Location\_id | Not null | Number(4) |
| St\_addr |  | Varchar(40) |
| Postal\_code |  | Varchar(12) |
| City | Not null | Varchar(30) |
| State\_province |  | Varchar(25) |
| Country\_id |  | Char(2) |

1. Create the DEPT table based on the DEPARTMENT following the table instance chart below. Confirm that the table is created.

| **Column name** | ID | NAME |
| --- | --- | --- |
| **Key Type** |  |  |
| **Nulls/Unique** |  |  |
| **FK table** |  |  |
| **FK column** |  |  |
| **Data Type** | Number | Varchar2 |
| **Length** | 7 | 25 |

create table DEPT(ID number(7),NAME varchar2(25));

Output:

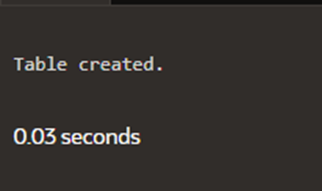


2. Create the EMP table based on the following instance chart. Confirm that the table is created.

| **Column name** | ID | LAST\_NAME | FIRST\_NAME | DEPT\_ID |
| --- | --- | --- | --- | --- |
| **Key Type** |  |  |  |  |
| **Nulls/Unique** |  |  |  |  |
| **FK table** |  |  |  |  |
| **FK column** |  |  |  |  |
| **Data Type** | Number | Varchar2 | Varchar2 | Number |
| **Length** | 7 | 25 | 25 | 7 |

create table emp(ID number(7),LAST\_NAME varchar2(25),FIRST\_NAME varchar2(25),DEPT\_ID number(7));

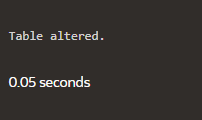
Output:



3. Modify the EMP table to allow for longer employee last names. Confirm the modification.(Hint: Increase the size to 50)

alter table emp modify LAST\_NAME varchar2(50);

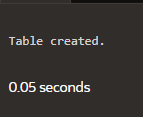
Output:



4. Create the EMPLOYEES2 table based on the structure of EMPLOYEES table. Include Only the Employee\_id, First\_name, Last\_name, Salary and Dept\_id coloumns. Name the columns Id, First\_name, Last\_name, salary and Dept\_id respectively.

create table EMPLOYEES2(Id number(7),Last\_name varchar2(25),First\_name varchar2(25),Dept\_id number(7),salary number(10));

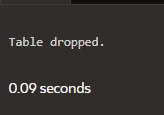
Output:



5. Drop the EMP table.

drop table EMP;

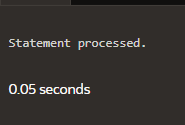
Output:



6. Rename the EMPLOYEES2 table as EMP.

rename EMPLOYEES2 to EMP;

Output:

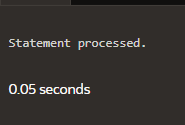


7. Add a comment on DEPT and EMP tables. Confirm the modification by describing the table.

comment on table DEPT is 'This is department table';

comment on table EMP is 'This is Employee table';

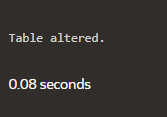
Output:



8. Drop the First\_name column from the EMP table and confirm it.

alter table EMP drop column First\_name;

Output:



| Evaluation Procedure | Marks awarded |
| --- | --- |
| Query(5) |  |
| Execution (5) |  |
| Viva(5) |  |
| Total (15) |  |
| Faculty Signature |  |

**EXERCISE-2**

**MANIPULATING DATA**

**OBJECTIVE**

After, the completion of this exercise the students will be able to do the following ● Describe each DML statement

● Insert rows into tables

● Update rows into table

● Delete rows from table

● Control Transactions

A DML statement is executed when you:

● Add new rows to a table

● Modify existing rows

● Removing existing rows

A transaction consists of a collection of DML statements that form a logical unit of work. **To Add a New Row**

INSERT Statement

**Syntax**

INSERT INTO table\_name VALUES (column1 values, column2 values, …, columnn values);

**Example:**

INSERT INTO department (70, ‘Public relations’, 100,1700);

**Inserting rows with null values**

**Implicit Method:** (Omit the column)

INSERT INTO department VALUES (30,’purchasing’);

**Explicit Method:** (Specify NULL keyword)

INSERT INTO department VALUES (100,’finance’, NULL, NULL);

**Inserting Special Values**

**Example:**

Using SYSDATE

INSERT INTO employees VALUES (113,’louis’, ‘popp’, ‘lpopp’,’5151244567’,**SYSDATE**, ‘ac\_account’, 6900, NULL, 205, 100);

**Inserting Specific Date Values**

**Example:**

INSERT INTO employees VALUES ( 114,’den’, ‘raphealy’, ‘drapheal’, ‘5151274561’, **TO\_DATE(‘feb 3,1999’,’mon, dd ,yyyy’),** ‘ac\_account’, 11000,100,30);

**To Insert Multiple Rows**

& is the placeholder for the variable value

**Example:**

INSERT INTO department VALUES (&dept\_id, &dept\_name, &location); **Copying Rows from another table**

⮚ Using Subquery

**Example:**

INSER INTO sales\_reps(id, name, salary, commission\_pct)

SELECt employee\_id, Last\_name, salary, commission\_pct

FROM employees

WHERE jod\_id LIKE ‘%REP’);

**CHANGING DATA IN A TABLE**

UPDATE Statement

**Syntax1: (** to update specific rows**)**

UPDATE table\_name SET column=value WHERE condition;

**Syntax 2:** (To updae all rows)

UPDATE table\_name SET column=value;

**Updating columns with a subquery**

UPDATE employees

SET job\_id= (SELECT job\_id

FROM employees

WHERE employee\_id=205)

WHERE employee\_id=114;

**REMOVING A ROW FROM A TABLE**

**DELETE STATEMENT**

**Syntax**

DELETE FROM table\_name WHERE conditions;

**Example:**

DELETE FROM department WHERE dept\_name=’finance’’;

**Find the Solution for the following:**

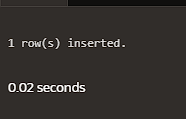
1. Create MY\_EMPLOYEE table with the following structure

| NAME | NULL? | TYPE |
| --- | --- | --- |
| ID | Not null | Number(4) |
| Last\_name |  | Varchar(25) |
| First\_name |  | Varchar(25) |
| Userid |  | Varchar(25) |
| Salary |  | Number(9,2) |

create table MY\_EMPLOYEE(ID number(4) not null,Last\_name varchar(25),Fisrt\_name

varchar(25),Userid varchar(25),Salary number(9,2));

Output:



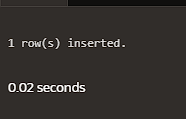
2. Add the first and second rows data to MY\_EMPLOYEE table from the following sample data.

| **ID** | **Last\_name** | **First\_name** | **Userid** | **salary** |
| --- | --- | --- | --- | --- |
| 1 | Patel | Ralph | rpatel | 895 |
| 2 | Dancs | Betty | bdancs | 860 |
| 3 | Biri | Ben | bbiri | 1100 |
| 4 | Newman | Chad | Cnewman | 750 |
| 5 | Ropebur | Audrey | aropebur | 1550 |

insert into MY\_EMPLOYEE values('1','Patel','Ralph','rpatel','895');

insert into MY\_EMPLOYEE values('2','Dancs','Betty','bdancs','860');

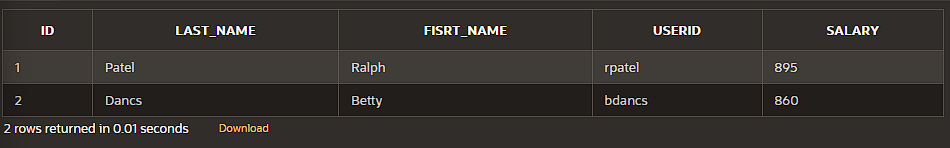
Output:



3. Display the table with values.

select \* from MY\_EMPLOYEE;

Output:

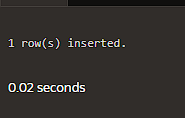


4. Populate the next two rows of data from the sample data. Concatenate the first letter of the first\_name with the first seven characters of the last\_name to produce Userid.

insert into MY\_EMPLOYEE values('3','Biri','Ben','bbiri','1100');

insert into MY\_EMPLOYEE values('4','Newman','Chad','Cnewman','750');

output:



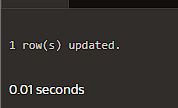
5. Make the data additions permanent.

Commit;

6. Change the last name of employee 3 to Drexler.

update MY\_EMPLOYEE set Last\_name='Drexler' where ID=3;

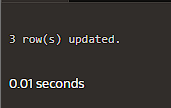
Output:



7. Change the salary to 1000 for all the employees with a salary less than 900.

update MY\_EMPLOYEE set salary='1000' where salary<900;

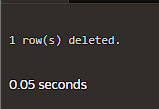
Output:



8. Delete Betty dancs from MY \_EMPLOYEE table.

delete from MY\_EMPLOYEE where First\_name='Betty' and Last\_name='Dancs';

Output:



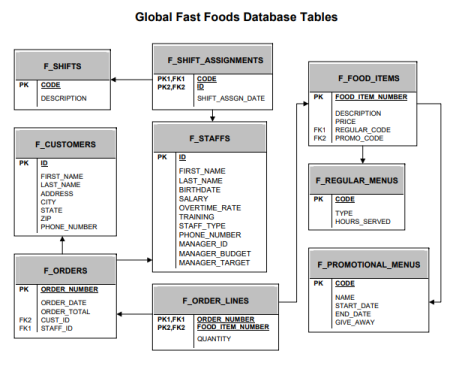
9. Empty the fourth row of the emp table.

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Query(5) |  |
| Execution (5) |  |
| Viva(5) |  |
| Total (15) |  |
| Faculty Signature |  |

**PRACTICE QUESTIONS**

**Date:**

**Working with Columns, Characters, and Rows**

****

1. The manager of Global Fast Foods would like to send out coupons for the upcoming sale. He wants to send one coupon to each household. Create the SELECT statement that returns the customer last name and a mailing address.

2. Each statement below has errors. Correct the errors and execute the query in Oracle Application Express.

a.

SELECT first name FROM

f\_staffs;

b.

SELECT first\_name |" " | last\_name AS "DJs on Demand Clients" FROM

d\_clients;

c.

SELECT DISCTINCT f\_order\_lines

FROM quantity;

d.

SELECT order number

FROM f\_orders;

3. Sue, Bob, and Monique were the employees of the month. Using the f\_staffs table, create a SELECT statement to display the results as shown in the Super Star chart.

| Super Star |
| --- |
| \*\*\* Sue \*\*\* Sue \*\*\* |
| \*\*\* Bob \*\*\* Bob \*\*\* |
| \*\*\* Monique \*\*\* Monique \*\*\* |

4.Which of the following is TRUE about the following query?

SELECT first\_name, DISTINCT birthdate

FROM f\_staffs;

a. Only two rows will be returned.

b. Four rows will be returned.

c. Only Fred 05-Jan-1988 and Lizzie 10-Nov-1987 will be returned.

d. No rows will be returned.

5. Global Fast Foods has decided to give all staff members a 5% raise. Prepare a report that presents the output as shown in the chart.

| **EMPLOYEE LAST NAME** | **CURRENT SALARY** | **SALARY WITH 5% RAISE** |
| --- | --- | --- |
|  |  |  |

6. Create a query that will return the structure of the Oracle database EMPLOYEES table. Which columns are marked “nullable”? What does this mean?

7. The owners of DJs on Demand would like a report of all items in their D\_CDs table with the following column headings: Inventory Item, CD Title, Music Producer, and Year Purchased. Prepare this report.

8.True/False – The following SELECT statement executes successfully: SELECT last\_name, job\_id, salary AS Sal FROM employees;

9.True/False – The following SELECT statement executes successfully: SELECT \* FROM job\_grades;

10.There are four coding errors in this statement. Can you identify them?

SELECT employee\_id, last\_name sal x 12 ANNUAL SALARY FROM employees; 11.In the arithmetic expression salary\*12 - 400, which operation will be evaluated first?

12. Which of the following can be used in the SELECT statement to return all columns of data in the Global Fast Foods f\_staffs table?

a. column names

b. \*

c. DISTINCT id

d. both a and b

13. Using SQL to choose the columns in a table uses which capability?

a. selection

b. projection

c. partitioning

d. join

14. SELECT last\_name AS "Employee". The column heading in the query result will appear as: a. EMPLOYEE

b. employee

c. Employee

d. "Employee:

15. Which expression below will produce the largest value?

a. SELECT salary\*6 + 100

b. SELECT salary\* (6 + 100)

c. SELECT 6(salary+ 100)

d. SELECT salary+6\*100

16. Which statement below will return a list of employees in the following format? Mr./Ms. Steven King is an employee of our company.

a. SELECT "Mr./Ms."||first\_name||' '||last\_name 'is an employee of our company.' AS "Employees" FROM employees;

b. SELECT 'Mr./Ms. 'first\_name,last\_name ||' '||'is an employee of our company.' FROM employees;

c. SELECT 'Mr./Ms. '||first\_name||' '||last\_name ||' '||'is an employee of our company.' AS "Employees" FROM employees ;

d. SELECT Mr./Ms. ||first\_name||' '||last\_name ||' '||"is an employee of our company." AS "Employees" FROM employees

17. Which is true about SQL statements?

a. SQL statements are case-sensitive

b. SQL clauses should not be written on separate lines.

c. Keywords cannot be abbreviated or split across lines.

d. SQL keywords are typically entered in lowercase; all other words in uppercase.

18. Which queries will return three columns each with UPPERCASE column headings? a. SELECT "Department\_id", "Last\_name", "First\_name"

FROM employees;

b. SELECT DEPARTMENT\_ID, LAST\_NAME, FIRST\_NAME

FROM employees;

c. SELECT department\_id, last\_name, first\_name AS UPPER CASE

FROM employees

d. SELECT department\_id, last\_name, first\_name

FROM employees;

19. Which statement below will likely fail?

a. SELCT \* FROM employees;

b. Select \* FROM employees;

c. SELECT \* FROM EMPLOYEES;

d. SelecT\* FROM employees;

20. Click on the History link at the bottom of the SQL Commands window. Scroll or use the arrows at the bottom of the page to find the statement you wrote to solve problem 3 above. (The one with the column heading SuperStar). Click on the statement to load it back into the command window. Execute the command again, just to make sure it is the correct one that works. Once you know it works, click on the SAVE button in the top right corner of the SQL Commands window, and enter a name for your saved statement. Use your own initials and “\_superstar.sql”, so if your initials are CT then the filename will be CT\_superstar.sql.

Log out of OAE, and log in again immediately. Navigate back to the SQL Commands window, click the Saved SQL link at the bottom of the page and load your saved SQL statement into the Edit window. This is done by clicking on the script name. Edit the statement, to make it display + instead of \*. Run your amended statement and save it as initials\_superplus.sql.

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Practice Evaluation (5) |  |
| Viva(5) |  |
| Total (10) |  |
| Faculty Signature |  |

**EXERCISE-3**

**INCLUDING CONSTRAINTS**

**OBJECTIVE**

After the completion of this exercise the students should be able to do the following ● Describe the constraints

● Create and maintain the constraints

**What are Integrity constraints?**

● Constraints enforce rules at the table level.

● Constraints prevent the deletion of a table if there are dependencies

**The following types of integrity constraints are valid**

a) **Domain Integrity**

✔ NOT NULL

✔ CHECK

b) **Entity Integrity**

✔ UNIQUE

✔ PRIMARY KEY

c) **Referential Integrity**

✔ FOREIGN KEY

**Constraints can be created in either of two ways**

1. At the same time as the table is created

2. After the table has been created.

**Defining Constraints**

Create table tablename (column\_name1 data\_ type constraints, column\_name2 data\_ type constraints …);

**Example:**

Create table employlees ( employee\_id number(6), first\_name varchar2(20), ..job\_id varchar2 (10), CONSTRAINT emp\_emp\_id\_pk PRIMARY KEY (employlee\_id));

**Domain Integrity**

This constraint sets a range and any violations that takes place will prevent the user from performing the manipulation that caused the breach.It includes:

**NOT NULL Constraint**

While creating tables, by default the rows can have null value.the enforcement of not null constraint in a table ensure that the table contains values.

**Principle of null values:**

o Setting null value is appropriate when the actual value is unknown, or when a value would not be meaningful.

o A null value is not equivalent to a value of zero.

o A null value will always evaluate to null in any expression.

o When a column name is defined as not null, that column becomes a mandatory i.e., the user has to enter data into it.

o Not null Integrity constraint cannot be defined using the alter table command when the table contain rows.

**Example**

CREATE TABLE employees (employee\_id number (6), last\_name varchar2(25) NOT NULL, salary number(8,2), commission\_pct number(2,2), hire\_date date constraint emp\_hire\_date\_nn NOT NULL’….);

**CHECK**

Check constraint can be defined to allow only a particular range of values.when the manipulation violates this constraint,the record will be rejected.Check condition cannot contain sub queries.

CREATE TABLE employees (employee\_id number (6), last\_name varchar2 (25) NOT NULL, salary number(8,2), commission\_pct number(2,2), hire\_date date constraint emp\_hire\_date\_nn NOT NULL’…,CONSTRAINT emp\_salary\_mi CHECK(salary > 0));

**Entity Integrity**

Maintains uniqueness in a record. An entity represents a table and each row of a table represents an instance of that entity. To identify each row in a table uniquely we need to use this constraint. There are 2 entity constraints:

**a) Unique key constraint**

It is used to ensure that information in the column for each record is unique, as with telephone or driver’s license numbers. It prevents the duplication of value with rows of a specified column in a set of column. A column defined with the constraint can allow null value.

If unique key constraint is defined in more than one column i.e., combination of column cannot be specified. Maximum combination of columns that a composite unique key can contain is 16.

**Example:**

CREATE TABLE employees (employee\_id number(6), last\_name varchar2(25) NOT NULL,email varchar2(25), salary number(8,2), commission\_pct number(2,2), hire\_date date constraint emp\_hire\_date\_nn NOT NULL’ COSTRAINT emp\_email\_uk UNIQUE(email));

**PRIMARY KEY CONSTRAINT**

A primary key avoids duplication of rows and does not allow null values. Can be defined on one or more columns in a table and is used to uniquely identify each row in a table. These values should never be changed and should never be null.

A table should have only one primary key. If a primary key constraint is assigned to more than one column or combination of column is said to be composite primary key, which can contain 16 columns.

**Example:**

CREATE TABLE employees (employee\_id number(6) , last\_name varchar2(25) NOT NULL,email varchar2(25), salary number(8,2), commission\_pct number(2,2), hire\_date date constraint emp\_hire\_date\_nn NOT NULL, Constraint emp\_id pk PRIMARY KEY (employee\_id),CONSTRAINT emp\_email\_uk UNIQUE(email));

**c) Referential Integrity**

It enforces relationship between tables. To establish parent-child relationship between 2 tables having a common column definition, we make use of this constraint. To implement this, we should define the column in the parent table as primary key and same column in the child table as foreign key referring to the corresponding parent entry.

**Foreign key**

A column or combination of column included in the definition of referential integrity, which would refer to a referenced key.

**Referenced key**

It is a unique or primary key upon which is defined on a column belonging to the parent table. Keywords:

**FOREIGN KEY:** Defines the column in the child table at the table level constraint. **REFERENCES:** Identifies the table and column in the parent table.

**ON DELETE CASCADE:** Deletes the dependent rows in the child table when a row in the parent table is deleted.

**ON DELETE SET NULL:** converts dependent foreign key values to null when the parent value is removed.

CREATE TABLE employees (employee\_id number(6) , last\_name varchar2(25) NOT NULL,email varchar2(25), salary number(8,2), commission\_pct number(2,2), hire\_date date constraint emp\_hire\_date\_nn NOT NULL, Constraint emp\_id pk PRIMARY KEY (employee\_id),CONSTRAINT emp\_email\_uk UNIQUE(email),CONSTRAINT emp\_dept\_fk FOREIGN KEY (department\_id) references deparments(dept\_id));

**ADDING A CONSTRAINT**

Use the ALTER to

● Add or Drop a constraint, but not modify the structure

● Enable or Disable the constraints

● Add a not null constraint by using the Modify clause

**Syntax**

ALTER TABLE table name ADD CONSTRAINT Cons\_name type(column name); **Example:**

ALTER TABLE employees ADD CONSTRAINT emp\_manager\_fk FOREIGN KEY (manager\_id) REFERENCES employees (employee\_id);

**DROPPING A CONSTRAINT**

**Example:**

ALTER TABLE employees DROP CONSTRAINT emp\_manager\_fk;

**CASCADE IN DROP**

● The CASCADE option of the DROP clause causes any dependent constraints also to be dropped.

**Syntax**

ALTER TABLE departments DROP PRIMARY KEY|UNIQUE (column)| CONSTRAINT constraint \_name CASCADE;

**DISABLING CONSTRAINTS**

● Execute the DISABLE clause of the ALTER TABLE statement to deactivate an integrity constraint

● Apply the CASCADE option to disable dependent integrity constraints. **Example**

ALTER TABLE employees DISABLE CONSTRAINT emp\_emp\_id\_pk CASCADE; **ENABLING CONSTRAINTS**

● Activate an integrity constraint currently disabled in the table definition by using the ENABLE clause.

**Example**

ALTER TABLE employees ENABLE CONSTRAINT emp\_emp\_id\_pk CASCADE; **CASCADING CONSTRAINTS**

The CASCADE CONSTRAINTS clause is used along with the DROP column clause. It drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped Columns.

This clause also drops all multicolumn constraints defined on the dropped column.

**Example:**

**Assume table TEST1 with the following structure**

CREATE TABLE test1 ( pk number PRIMARY KEY, fk number, col1 number,col2 number, CONTRAINT fk\_constraint FOREIGN KEY(fk) references test1, CONSTRAINT ck1 CHECK (pk>0 and col1>0), CONSTRAINT ck2 CHECK (col2>0));

**An error is returned for the following statements**

ALTER TABLE test1 DROP (pk);

ALTER TABLE test1 DROP (col1);

**The above statement can be written with CASCADE CONSTRAINT**

ALTER TABLE test 1 DROP(pk) CASCADE CONSTRAINTS;

**(OR)**

ALTER TABLE test 1 DROP(pk, fk, col1) CASCADE CONSTRAINTS;

**VIEWING CONSTRAINTS**

Query the USER\_CONSTRAINTS table to view all the constraints definition and names. **Example:**

SELECT constraint\_name, constraint\_type, search\_condition FROM user\_constraints WHERE table\_name=’employees’;

**Viewing the columns associated with constraints**

SELECT constraint\_name, constraint\_type, FROM user\_cons\_columns

WHERE table\_name=’employees’;

**Find the Solution for the following:**

1. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column.The constraint should be named at creation. Name the constraint my\_emp\_id\_pk.

2. Create a PRIMAY KEY constraint to the DEPT table using the ID colum. The constraint should be named at creation. Name the constraint my\_dept\_id\_pk.

3. Add a column DEPT\_ID to the EMP table. Add a foreign key reference on the EMP table that ensures that the employee is not assigned to nonexistent deparment. Name the constraint my\_emp\_dept\_id\_fk.

4. Modify the EMP table. Add a COMMISSION column of NUMBER data type, precision 2, scale 2. Add a constraint to the commission column that ensures that a commission value is greater than zero.

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Query(5) |  |
| Execution (5) |  |
| Viva(5) |  |
| Total (15) |  |
| Faculty Signature |  |

**PRACTICE QUESTIONS**

**Limit Rows Selected**

1. Using the Global Fast Foods database, retrieve the customer’s first name, last name, and address for the customer who uses ID 456.

2. Show the name, start date, and end date for Global Fast Foods' promotional item “ballpen and highlighter” giveaway.

3. Create a SQL statement that produces the following output:

Oldest

The 1997 recording in our database is The Celebrants Live in Concert

4. The following query was supposed to return the CD title "Carpe Diem" but no rows were returned. Correct the mistake in the statement and show the output.

SELECT produce, title

FROM d\_cds

WHERE title = 'carpe diem' ;

5. The manager of DJs on Demand would like a report of all the CD titles and years of CDs that were produced before 2000.

6. Which values will be selected in the following query?

SELECT salary

FROM employees

WHERE salary < = 5000;

a. 5000

b. 0 - 4999

c. 2500

d. 5

7. Write a SQL statement that will display the student number (studentno), first name (fname), and last name (lname) for all students who are female (F) in the table named students.

8. Write a SQL statement that will display the student number (studentno) of any student who has a PE major in the table named students. Title the studentno column Student Number.

9. Write a SQL statement that lists all information about all male students in the table named students.

10. Write a SQL statement that will list the titles and years of all the DJs on Demand's CDs that were not produced in 2000.

11. Write a SQL statement that lists the Global Fast Foods employees who were born before 1980.

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Practice Evaluation (5) |  |
| Viva(5) |  |
| Total (10) |  |
| Faculty Signature |  |

**EXERCISE-4**

**Writing Basic SQL SELECT Statements**

**OBJECTIVES**

After the completion of this exercise, the students will be able to do the following: ● List the capabilities of SQL SELECT Statement

● Execute a basic SELECT statement

**Capabilities of SQL SELECT statement**

A SELECT statement retrieves information from the database. Using a select statement, we can perform

✔ Projection: To choose the columns in a table

✔ Selection: To choose the rows in a table

✔ Joining: To bring together the data that is stored in different tables **Basic SELECT Statement**

**Syntax**

SELECT \*|DISTINCT Column\_ name| alias

` FROM table\_name;

**NOTE:**

DISTINCT—Suppr

ess the duplicates.

Alias—gives selected columns different headings.

**Example: 1**

SELECT \* FROM departments;

**Example: 2**

SELECT location\_id, department\_id FROM departments;

**Writing SQL Statements**

● SQL statements are not case sensitive

● SQL statements can be on one or more lines.

● Keywords cannot be abbreviated or split across lines

● Clauses are usually placed on separate lines

● Indents are sued to enhance readability

**Using Arithmetic Expressions**

Basic Arithmetic operators like \*, /, +, -can be used

**Example:1**

SELECT last\_name, salary, salary+300 FROM employees;

**Example:2**

SELECT last\_name, salary, 12\*salary+100 FROM employees;

The statement is not same as

SELECT last\_name, salary, 12\*(salary+100) FROM employees;

**Example:3**

SELECT last\_name, job\_id, salary, commission\_pct FROM employees; **Example:4**

SELECT last\_name, job\_id, salary, 12\*salary\*commission\_pct FROM employees; **Using Column Alias**

● To rename a column heading with or without AS keyword.

**Example:1**

SELECT last\_name AS Name

FROM employees;

**Example: 2**

SELECT last\_name “Name” salary\*12 “Annual Salary “

FROM employees;

**Concatenation Operator**

● Concatenates columns or character strings to other columns ● Represented by two vertical bars (||)

● Creates a resultant column that is a character expression

**Example:**

SELECT last\_name||job\_id AS “EMPLOYEES JOB” FROM employees; **Using Literal Character String**

● A literal is a character, a number, or a date included in the SELECT list. ● Date and character literal values must be enclosed within single quotation marks.

**Example:**

SELECT last\_name||’is a’||job\_id AS “EMPLOYEES JOB” FROM employees; **Eliminating Duplicate Rows**

● Using DISTINCT keyword.

**Example:**

SELECT DISTINCT deparment\_id FROM employees;

**Displaying Table Structure**

● Using DESC keyword.

**Syntax**

DESC table\_name;

**Example:**

DESC employees;

**Find the Solution for the following:**

**True OR False**

1. The following statement executes successfully.

**Identify the Errors**

SELECT employee\_id, last\_name

sal\*12 ANNUAL SALARY

FROM employees;

**Queries**

2. Show the structure of departments the table. Select all the data from it.

3. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first.

4. Provide an alias STARTDATE for the hire date.

5. Create a query to display unique job codes from the employee table.

6. Display the last name concatenated with the job ID , separated by a comma and space, and name the column EMPLOYEE and TITLE.

7. Create a query to display all the data from the employees table. Separate each column by a comma. Name the column THE\_OUTPUT.

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Query(5) |  |
| Execution (5) |  |
| Viva(5) |  |
| Total (15) |  |
| Faculty Signature |  |

**Practice Questions**

**COMPARISON OPERATORS**

1. Who are the partners of DJs on Demand who do not get an authorized expense amount?

2. Select all the Oracle database employees whose last names end with “s”. Change the heading of the column to read Possible Candidates.

3. Which statement(s) are valid?

a. WHERE quantity <> NULL;

b. WHERE quantity = NULL;

c. WHERE quantity IS NULL;

d. WHERE quantity != NULL;

4. Write a SQL statement that lists the songs in the DJs on Demand inventory that are type code 77, 12, or 1.

**Logical Comparisons and Precedence Rules**

1. Execute the two queries below. Why do these nearly identical statements produce two different results? Name the difference and explain why.

SELECT code, description

FROM d\_themes

WHERE code >200 AND description IN('Tropical', 'Football', 'Carnival'); SELECT code, description

FROM d\_themes

WHERE code >200 OR description IN('Tropical', 'Football', 'Carnival');

2. Display the last names of all Global Fast Foods employees who have “e” and “i” in their last names.

3. “I need to know who the Global Fast Foods employees are that make more than $6.50/hour and their position is not order taker.”

4. Using the employees table, write a query to display all employees whose last names start with “D” and have “a” and “e” anywhere in their last name.

5. In which venues did DJs on Demand have events that were not in private homes?

6. Which list of operators is in the correct order from highest precedence to lowest precedence? a. AND, NOT, OR

b. NOT, OR, AND

c. NOT, AND, OR

**For questions 7 and 8, write SQL statements that will produce the desired output.**

7. Who am I?

I was hired by Oracle after May 1998 but before June of 1999. My salary is less than $8000 per month, and I have an “en” in my last name.

8. What's my email address?

Because I have been working for Oracle since the beginning of 1996, I make more than $9000 per month. Because I make so much money, I don't get a commission

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Practice Evaluation (5) |  |
| Viva(5) |  |
| Total (10) |  |
| Faculty Signature |  |

**EXERCISE-5**

**Restricting and Sorting data**

After the completion of this exercise, the students will be able to do the following: ● Limit the rows retrieved by the queries

● Sort the rows retrieved by the queries

●

**Limiting the Rows selected**

● Using WHERE clause

● Alias cannot used in WHERE clause

**Syntax**

SELECT----------

FROM----------

WHERE condition;

**Example:**

SELECT employee\_id,last\_name, job\_id, deparment\_id FROM employees WHERE department\_id=90;

**Character strings and Dates**

Character strings and date values are enclosed in single quotation marks. Character values are case sensitive and date values are format sensitive.

**Example:**

SELECT employee\_id,last\_name, job\_id, deparment\_id FROM employees WHERE last\_name=’WHALEN”;

**Comparison Conditions**

All relational operators can be used. (=, >, >=, <, <= ,<>,!=)

**Example:**

SELECT last\_name, salary

FROM employees

WHERE salary<=3000;

**Other comparison conditions**

| Operator | Meaning |
| --- | --- |
| BETWEEN …AND… | Between two values |
| IN | Match any of a list of values |
| LIKE | Match a character pattern |
| IS NULL | Is a null values |

**Example:1**

SELECT last\_name, salary

FROM employees

WHERE salary BETWEEN 2500 AND 3500;

**Example:2**

SELECT employee\_id, last\_name, salary , manager\_id

FROM employees

WHERE manager\_id IN (101, 100,201);

**Example:3**

● Use the LIKE condition to perform wildcard searches of valid string values. ● Two symbols can be used to construct the search string

▪ % denotes zero or more characters

▪ \_ denotes one character

SELECT first\_name, salary

FROM employees

WHERE first\_name LIKE ‘%s’;

**Example:4**

SELECT last\_name, salary

FROM employees

WHERE last\_name LIKE ‘\_o%’;

**Example:5**

**ESCAPE option**-To have an exact match for the actual % and\_ characters To search for the string that contain ‘SA\_’

SELECT employee\_id, first\_name, salary,job\_id

FROM employees

WHERE job\_id LIKE ‘%sa\\_%’ESCAPE’\’;

**Test for NULL**

● Using IS NULL operator

**Example:**

SELECT employee\_id, last\_name, salary , manager\_id

FROM employees

WHERE manager\_id IS NULL;

**Logical Conditions**

All logical operators can be used.( AND,OR,NOT)

**Example:1**

SELECT employee\_id, last\_name, salary , job\_id

FROM employees

WHERE salary>=10000

AND job\_id LIKE ‘%MAN%’;

**Example:2**

SELECT employee\_id, last\_name, salary , job\_id

FROM employees

WHERE salary>=10000

OR job\_id LIKE ‘%MAN%’;

**Example:3**

SELECT employee\_id, last\_name, salary , job\_id

FROM employees

WHERE job\_id NOT IN (‘it\_prog’, st\_clerk’, sa\_rep’); **Rules of Precedence**

| **Order**  **Evaluated** | **Operator** |
| --- | --- |
| 1 | Arithmetic |
| 2 | Concatenation |
| 3 | Comparison |
| 4 | IS [NOT] NULL, LIKE, [NOT] IN |
| 5 | [NOT] BETWEEN |
| 6 | Logical NOT |
| 7 | Logical AND |
| 8 | Logical OR |

**Example:1**

SELECT employee\_id, last\_name, salary , job\_id

FROM employees

WHERE job\_id =’sa\_rep’

OR job\_id=’ad\_pres’

AND salary>15000;

**Example:2**

SELECT employee\_id, last\_name, salary , job\_id

FROM employees

WHERE (job\_id =’sa\_rep’

OR job\_id=’ad\_pres’)

AND salary>15000;

**Sorting the rows**

Using ORDER BY Clause

**ASC**-Ascending Order,Default

**DESC**-Descending order

**Example:1**

SELECT last\_name, salary , job\_id,department\_id,hire\_date

FROM employees

ORDER BY hire\_date;

**Example:2**

SELECT last\_name, salary , job\_id,department\_id,hire\_date

FROM employees

ORDER BY hire\_date DESC;

**Example:3**

**Sorting by column alias**

SELECT last\_name, salary\*12 annsal , job\_id,department\_id,hire\_date

FROM employees

ORDER BY annsal;

**Example:4**

**Sorting by Multiple columns**

SELECT last\_name, salary , job\_id,department\_id,hire\_date

FROM employees

ORDER BY department\_id, salary DESC;

**Find the Solution for the following:**

1. Create a query to display the last name and salary of employees earning more than 12000.

2. Create a query to display the employee last name and department number for employee number 176.

3. Create a query to display the last name and salary of employees whose salary is not in the range of 5000 and 12000. (hints: not between )

4. Display the employee last name, job ID, and start date of employees hired between February 20,1998 and May 1,1998.order the query in ascending order by start date.(hints: between)

5. Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.(hints: in, orderby)

6. Display the last name and salary of all employees who earn between 5000 and 12000 and are in departments 20 and 50 in alphabetical order by name. Label the columns EMPLOYEE, MONTHLY SALARY respectively.(hints: between, in)

7. Display the last name and hire date of every employee who was hired in 1994.(hints: like)

8. Display the last name and job title of all employees who do not have a manager.(hints: is null)

9. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.(hints: is not nul,orderby)

10. Display the last name of all employees where the third letter of the name is ***a***.(hints:like)

11. Display the last name of all employees who have an a and an ***e*** in their last name.(hints: like)

12. Display the last name and job and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to 2500 ,3500 or 7000.(hints:in,not in)

13. Display the last name, salary, and commission for all employees whose commission amount is 20%.(hints:use predicate logic)

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Query(5) |  |
| Execution (5) |  |
| Viva(5) |  |
| Total (15) |  |
| Faculty Signature |  |

**Pracice Questions**

**Sorting Rows**

1. In the example below, assign the employee\_id column the alias of “Number.” Complete the SQL statement to order the result set by the column alias.

SELECT employee\_id, first\_name, last\_name FROM employees;

2. Create a query that will return all the DJs on Demand CD titles ordered by year with titles in alphabetical order by year.

3. Order the DJs on Demand songs by descending title. Use the alias "Our Collection" for the song title.

4. Write a SQL statement using the ORDER BY clause that could retrieve the information needed.

**EXERCISE-6**

**Single Row Functions Objective**

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Practice Evaluation (5) |  |
| Viva(5) |  |
| Total (10) |  |
| Faculty Signature |  |

After the completion of this exercise, the students will be able to do the following: ● Describe various types of functions available in SQL.

● Use character, number and date functions in SELECT statement.

● Describe the use of conversion functions. **Single row functions:**

Manipulate data items.

Accept arguments and return one value.

Act on each row returned.

Return one result per row.

May modify the data type.

Can be nested.

Accept arguments which can be a column or an expression

**Syntax**

Function\_name(arg1,…argn)

An argument can be one of the following

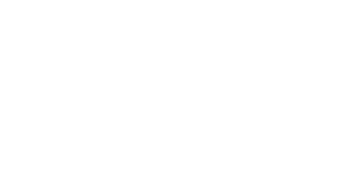
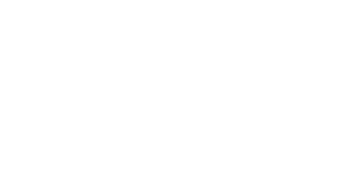
✔ User-supplied constant

✔ Variable value

✔ Column name

✔ Expression

**CHARACTER**

**GENERAL**

**CONVERSIO** 

**N**

**SINGLE-ROW FUNCTIONS** 

**NUMBER DATE**

● 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 data type.

● Conversion Functions: Convert a value from one type to another.

**Character Functions**

Character Functions

Case-manipulation functions Character-manipulation functions 1. Lower 1. Concat

2. Upper 2. Substr

3. Initcap 3. Length

4. Instr

5. Lpad/Rpad

6. Trim

7. Repalce

| **Function** | **Purpose** |
| --- | --- |
| lower(column/expr) | Converts alpha character values to lowercase |
| upper(column/expr) | Converts alpha character values to uppercase |
| initcap(column/expr) | Converts alpha character values the to uppercase for the first letter of each word, all other letters in lowercase |
| concat(column1/expr1, column2/expr2) | Concatenates the first character to the second character |
| substr(column/expr,m,n) | Returns specified characters from character value starting at character position m, n characters long |
| length(column/expr) | Returns the number of characters in the expression |
| instr(column/expr,’string’,m,n) | Returns the numeric position of a named string |
| lpad(column/expr, n,’string’) | Pads the character value right-justified to a total width of n character positions |
| rpad(column/expr,’string’,m,n) | Pads the character value left-justified to a total width of n character positions |
| trim(leading/trailing/both, trim\_character FROM trim\_source) | Enables you to trim heading or string. trailing or both from a character |
| replace(text, search\_string,  replacement\_string) |  |

**Example:**

lower(‘SQL Course’)��sql course

upper(‘SQL Course’)��SQL COURSE

initcap(‘SQL Course’)��Sql Course

SELECT ‘The job id for’|| upper(last\_name||’is’||lower(job\_id) AS “EMPLOYEE DETAILS” FROM employees;

SELECT employee\_id, last\_name, department\_id

FROM employees

WHERE LOWER(last\_name)=’higgins’;

| **Function** | **Result** |
| --- | --- |
| CONCAT(‘hello’, ‘world’) | helloworld |
| Substr(‘helloworld’,1,5) | Hello |
| Length(‘helloworld’) | 10 |

| Instr(‘helloworld’,’w’) | 6 |
| --- | --- |
| Lpad(salary,10,’\*’) | \*\*\*\*\*24000 |
| Rpad(salary,10,’\*’) | 24000\*\*\*\*\* |
| Trim(‘h’ FROM ‘helloworld’) | elloworld |

| **Command** | **Query** | **Output** |
| --- | --- | --- |
| initcap(char); | *select initcap(“hello”) from dual;* | Hello |
| lower (char);  upper (char); | *select lower (‘HELLO’) from dual;*  *select upper (‘hello’) from dual;* | Hello  HELLO |
| ltrim (char,[set]); | *select ltrim (‘cseit’, ‘cse’) from dual;* | IT |
| rtrim (char,[set]); | *select rtrim (‘cseit’, ‘it’) from dual;* | CSE |
| replace (char,search string, replace string); | *select replace (‘jack and jue’, ‘j’, ‘bl’) from dual;* | black and blue |
| substr (char,m,n); | *select substr (‘information’, 3, 4) from dual;* | form |

**Example:**

SELECT employee\_id, CONCAT (first\_name,last\_name) NAME , job\_id,LENGTH(last\_name), INSTR(last\_name,’a’) “contains’a’?”

FROM employees WHERE SUBSTR(job\_id,4)=’ERP’;

**NUMBER FUNCTIONS**

| **Function** | **Purpose** |
| --- | --- |
| round(column/expr, n) | Rounds the value to specified decimal |
| trunc(column/expr,n) | Truncates value to specified decimal |
| mod(m,n) | Returns remainder of division |

**Example**

| Function | Result |
| --- | --- |
| round(45.926,2) | 45.93 |
| trunc(45.926,2) | 45.92 |
| mod(1600,300) | 100 |

SELECT ROUND(45.923,2), ROUND(45.923,0), ROUND(45.923,-1) FROM dual; **NOTE:** Dual is a dummy table you can use to view results from functions and calculations. SELECT TRUNC(45.923,2), TRUNC(45.923), TRUNC(45.923,-2) FROM dual;

SELECT last\_name,salary,MOD(salary,5000) FROM employees WHERE job\_id=’sa\_rep’; **Working with Dates**

The Oracle database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds.

• The default date display format is DD-MON-RR.

– Enables you to store 21st-century dates in the 20th century by specifying only the last two digits of the year

– Enables you to store 20th-century dates in the 21st century in the same way **Example**

SELECT last\_name, hire\_date FROM employees WHERE hire\_date < '01-FEB-88; **Working with Dates**

SYSDATE is a function that returns:

• Date

• Time

**Example**

**Display the current date using the DUAL table.**

SELECT SYSDATE FROM DUAL;

**Arithmetic with Dates**

• Add or subtract a number to or from a date for a resultant date value.

• Subtract two dates to find the number of days between those dates.

• Add hours to a date by dividing the number of hours by 24.

**Arithmetic with Dates**

Because 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:

**Operation Result Description**

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

**Example**

SELECT last\_name, (SYSDATE-hire\_date)/7 AS WEEKS

FROM employees

WHERE department\_id = 90;

**Date Functions**

****

**Date Functions**

Date functions operate on Oracle dates. All date functions return a value of DATE data type 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 that is 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 that is specified by the format model fmt. If the format model fmt is omitted, date is truncated to the nearest day.

**Using Date Functions**

****

**Example**

Display the employee number, hire date, number of months employed, sixmonth review date, first Friday after hire date, and last day of the hire month for all employees who have been employed for fewer than 70 months.

SELECT employee\_id, hire\_date,MONTHS\_BETWEEN (SYSDATE, hire\_date) TENURE,ADD\_MONTHS (hire\_date, 6) REVIEW,NEXT\_DAY (hire\_date, 'FRIDAY'), LAST\_DAY(hire\_date)

FROM employees

WHERE MONTHS\_BETWEEN (SYSDATE, hire\_date) < 70;

**Conversion Functions**

This covers the following topics:

• Writing a query that displays the current date

• Creating queries that require the use of numeric, character, and date functions • Performing calculations of years and months of service for an employee



**Implicit Data Type Conversion**

For assignments, the Oracle server can automatically convert the following: 

For example, the expression hire\_date > '01-JAN-90' results in the implicit conversion from the string '01-JAN-90' to a date.

For expression evaluation, the Oracle Server can automatically convert the following: 

**Explicit Data Type Conversion**

****

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

**Using the TO\_CHAR Function with Dates**

TO\_CHAR(date, 'format\_model')

**The format model:**

• Must be enclosed by single quotation marks

• Is case-sensitive

• Can include any valid date format element

• Has an fm element to remove padded blanks or suppress leading zeros

• Is separated from the date value by a comma

SELECT employee\_id, TO\_CHAR(hire\_date, 'MM/YY') Month\_Hired

FROM employees WHERE last\_name = 'Higgins';

**Elements of the Date Format Model**

****

**Sample Format Elements of Valid Date**

**Date Format Elements:** Time Formats

Use the formats that are listed in the following tables to display time information and literals and to change numerals to spelled numbers.



**Example**

SELECT last\_name,

TO\_CHAR(hire\_date, 'fmDD Month YYYY') AS HIREDATE

FROM employees;

Modify example to display the dates in a format that appears as “Seventeenth of June 1987 12:00:00 AM.”

SELECT last\_name,

TO\_CHAR (hire\_date, 'fmDdspth "of" Month YYYY fmHH:MI:SS AM') HIREDATE FROM employees;

**Using the TO\_CHAR Function with Numbers**

TO\_CHAR(number, 'format\_model')

These are some of the format elements that you can use with the TO\_CHAR function to display a number value as a character:



**Number Format Elements**

If you are converting a number to the character data type, you can use the following format elements:



SELECT TO\_CHAR(salary, '$99,999.00') SALARY

FROM employees

WHERE last\_name = 'Ernst';

**Using the TO\_NUMBER and TO\_DATE Functions**

• Convert a character string to a number format using the TO\_NUMBER function: TO\_NUMBER(char[, 'format\_model']

• Convert a character string to a date format using the TO\_DATE function:

TO\_DATE(char[, 'format\_model']

• These functions have an fx modifier. This modifier 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 a TO\_DATE function:

• Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model.

• The character argument cannot have extra blanks. Without fx, Oracle ignores extra blanks.

• Numeric data in the character argument must have the same number of digits as the corresponding element in the format model. Without fx, numbers in the character argument can omit leading zeros.

SELECT last\_name, hire\_date

FROM employees

WHERE hire\_date = TO\_DATE('May 24, 1999', 'fxMonth DD, YYYY');

**Find the Solution for the following:**

1. Write a query to display the current date. Label the column Date.

2. The HR department needs a report to display the employee number, last name, salary, and increased by 15.5% (expressed as a whole number) for each employee. Label the column New Salary.

3. Modify your query lab\_03\_02.sql to add a column that subtracts the old salary from the new salary. Label the column Increase.

4. Write a query that displays the last name (with the first letter uppercase and all other letters lowercase) and the length of the last name for all employees whose name starts with the letters J, A, or M. Give each column an appropriate label. Sort the results by the employees’ last names.

5. Rewrite the query so that the user is prompted to enter a letter that starts the last name. For example, if the user enters H when prompted for a letter, then the output should show all employees whose last name starts with the letter H.

6. The HR department wants to find the length of employment for each employee. For each employee, display the last name and calculate the number of months between today and the date on which 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.

**Note:** Your results will differ.

7. Create a report that produces the following for each employee:

<employee last name> earns <salary> monthly but wants <3 times salary>. Label the column Dream Salaries.

8. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the $ symbol. Label the column SALARY.

9. Display each employee’s last 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 “Monday, the Thirty-First of July, 2000.”

10. Display the last name, hire date, and day of the week on which the employee started. Label the column DAY. Order the results by the day of the week, starting with Monday.

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Query(5) |  |
| Execution (5) |  |
| Viva(5) |  |
| Total (15) |  |
| Faculty Signature |  |

**Practice Questions**

**Introduction to Functions**

1. For each task, choose whether a single-row or multiple row function would be most appropriate: a. Showing all of the email addresses in upper case letters

b. Determining the average salary for the employees in the sales department c. Showing hire dates with the month spelled out *(September 1, 2004)*

d. Finding out the employees in each department that had the most seniority (the earliest hire date) e. Displaying the employees’ salaries rounded to the hundreds place

f. Substituting zeros for null values when displaying employee commissions.

2. The most common multiple-row functions are: AVG, COUNT, MAX, MIN, and SUM. Give your own definition for each of these functions.

3. Test your definitions by substituting each of the multiple-row functions in this query. SELECT FUNCTION(salary)

FROM employees

Write out each query and its results.

**Case and Character Manipulation**

1. Using the three separate words “Oracle,” “Internet,” and

“Academy,” use one command to produce the following output:

The Best Class Oracle Internet Academy

2. Use the string “Oracle Internet Academy” to produce the following output: The Net net

3. What is the length of the string “Oracle Internet Academy”?

4. What’s the position of “I” in “Oracle Internet Academy”?

5. Starting with the string “Oracle Internet Academy”, pad the string to create \*\*\*\*Oracle\*\*\*\*Internet\*\*\*\*Academy\*\*\*\*

**Number Functions**

1. Display Oracle database employee last\_name and salary for employee\_ids between 100 and 102. Include a third column that divides each salary by 1.55 and rounds the result to two decimal places.

2. Display employee last\_name and salary for those employees who work in department 80. Give each of them a raise of 5.333% and truncate the result to two decimal places.

3. Use a MOD number function to determine whether 38873 is an even number or an odd number.

4. Use the DUAL table to process the following numbers:

845.553 - round to one decimal place

30695.348 - round to two decimal places

30695.348 - round to -2 decimal Places

2.3454 - truncate the 454 from the decimal

place

5. Divide each employee’s salary by 3. Display only those employees’ last names and salaries who earn a salary that is a multiple of 3.

6. Divide 34 by 8. Show only the remainder of the division. Name the output as EXAMPLE.

7. How would you like your paycheck – rounded or truncated? What if your paycheck was calculated to be $565.784 for the week, but you noticed that it was issued for $565.78. The loss of .004 cent would probably make very little difference to you. However, what if this was done to a thousand people, a 100,000 people, or a million people! Would it make a difference then? How much difference?

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Practice Evaluation (5) |  |
| Viva(5) |  |
| Total (10) |  |
| Faculty Signature |  |

**EXERCISE-7**

**Displaying data from multiple tables**

**Objective**

After the completion of this exercise, the students will be able to do the following: • Write SELECT statements to access data from more than one table using equality and nonequality joins

• View data that generally does not meet a join condition by using outer joins • Join a table to itself by using a self join

Sometimes you need to use data from more than one table.

**Cartesian Products**

• A Cartesian product is formed when:

– A join condition is omitted

– A join condition is invalid

– All rows in the first table are joined to all rows in the second table

• To avoid a Cartesian product, always include a valid join condition in a WHERE clause. A Cartesian product tends to generate a large number of rows, and the result is rarely useful. You should always include a valid join condition in a WHERE clause, unless you have a specific need to combine all rows from all tables.

Cartesian products are useful for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.

**Example:**

To displays employee last name and department name from the EMPLOYEES and DEPARTMENTS tables.

SELECT last\_name, department\_name dept\_name

FROM employees, departments;

**Types of Joins**

• Equijoin

• Non-equijoin

• Outer join

• Self join

• Cross joins

• Natural joins

• Using clause

• Full or two sided outer joins

• Arbitrary join conditions for outer joins

**Joining Tables Using Oracle Syntax**

SELECT table1.column, table2.column

FROM table1, table2

WHERE table1.column1 = table2.column2;

Write the join condition in the WHERE clause.

• Prefix the column name with the table name when the same column name appears in more than one table.

**Guidelines**

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

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

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

**What is an Equijoin?**

To determine an employee’s department name, you compare the value in the DEPARTMENT\_ID column in the EMPLOYEES table with the DEPARTMENT\_ID values in the DEPARTMENTS table.

The relationship between the EMPLOYEES and DEPARTMENTS tables is an equijoin—that is, values

in the DEPARTMENT\_ID column on both tables must be equal. Frequently, this type of join involves

primary and foreign key complements.

Note: Equijoins are also called simple joins or inner joins

SELECTemployees.employee\_id,employees.last\_name,employees.department\_id, departments.department\_id,departments.location\_id

FROM employees, departments

WHERE employees.department\_id = departments.department\_id;

**Additional Search Conditions**

**Using the AND Operator**

**Example:**

To display employee Matos’department number and department name, you need an additional condition in the WHERE clause.

SELECT last\_name, employees.department\_id,

department\_name

FROM employees, departments

WHERE employees.department\_id = departments.department\_id AND last\_name = ’Matos’; **Qualifying Ambiguous**

**Column Names**

• Use table prefixes to qualify column names that are in multiple tables.

• Improve performance by using table prefixes.

• Distinguish columns that have identical names but reside in different tables by using column aliases.

**Using Table Aliases**

• Simplify queries by using table aliases.

• Improve performance by using table prefixes

**Example:**

SELECT e.employee\_id, e.last\_name, e.department\_id,

d.department\_id, d.location\_id

FROM employees e , departments d

WHERE e.department\_id = d.department\_id;

**Joining More than Two Tables**

To join n tables together, you need a minimum of n-1 join conditions. For example, to join three

tables, a minimum of two joins is required.

**Example:**

To display the last name, the department name, and the city for each employee, you have to join the EMPLOYEES, DEPARTMENTS, and LOCATIONS tables.

SELECT e.last\_name, d.department\_name, l.city

FROM employees e, departments d, locations l

WHERE e.department\_id = d.department\_id

AND d.location\_id = l.location\_id;

**Non-Equijoins**

A non-equijoin is a join condition containing something other than an equality operator.The relationship between the EMPLOYEES table and the JOB\_GRADES table has an example of a non equijoin. A relationship between the two tables is that the SALARY column in the EMPLOYEES table must be between the values in the LOWEST\_SALARY and HIGHEST\_SALARY columns of

the JOB\_GRADES table. The relationship is obtained using an operator other than equals (=). **Example:**

SELECT e.last\_name, e.salary, j.grade\_level

FROM employees e, job\_grades j

WHERE e.salary

BETWEEN j.lowest\_sal AND j.highest\_sal;

**Outer Joins**

**Syntax**

• You use an outer join to also see rows that do not meet the join condition.

• The Outer join operator is the plus sign (+).

SELECT table1.column, table2.column

FROM table1, table2

WHERE table1.column(+) = table2.column;

SELECT table1.column, table2.column

FROM table1, table2

WHERE table1.column = table2.column(+);

The missing rows can be returned if an outer join operator is used in the join condition. The operator is a plus sign enclosed in parentheses (+), and it is placed on the “side” of the join that is deficient in information. This operator has the effect of creating one or more null rows, to which one or more rows

from the nondeficient table can be joined.

**Example:**

SELECT e.last\_name, e.department\_id, d.department\_name

FROM employees e, departments d

WHERE e.department\_id(+) = d.department\_id ;

**Outer Join Restrictions**

• The outer join operator can appear on only one side of the expression—the side that has information missing. It returns those rows from one table that have no direct match in the other table. • A condition involving an outer join cannot use the IN operator or be linked to another condition by the OR operator

**Self Join**

Sometimes you need to join a table to itself.

**Example:**

To find the name of each employee’s manager, you need to join the EMPLOYEES table to itself, or perform a self join.

SELECT worker.last\_name || ’ works for ’

|| manager.last\_name

FROM employees worker, employees manager

WHERE worker.manager\_id = manager.employee\_id ;

**Use a join to query data from more than one table.**

SELECT table1.column, table2.column

FROM table1

[CROSS JOIN table2] |

[NATURAL JOIN table2] |

[JOIN table2 USING (column\_name)] |

[JOIN table2

ON(table1.column\_name = table2.column\_name)] |

[LEFT|RIGHT|FULL OUTER JOIN table2

ON (table1.column\_name = table2.column\_name)];

In the syntax:

table1.column Denotes the table and column from which data is retrieved

CROSS JOIN Returns a Cartesian product from the two tables

NATURAL JOIN Joins two tables based on the same column name

JOIN table USING column\_name Performs an equijoin based on the column name JOIN table ON table1.column\_name Performs an equijoin based on the condition in the ON clause = table2.column\_name

**LEFT/RIGHT/FULL OUTER**

**Creating Cross Joins**

• The CROSS JOIN clause produces the crossproduct of two tables.

• This is the same as a Cartesian product between the two tables.

**Example:**

SELECT last\_name, department\_name

FROM employees

CROSS JOIN departments ;

SELECT last\_name, department\_name

FROM employees, departments;

**Creating Natural Joins**

• The NATURAL JOIN clause is based on all columns in the two tables that have the same name. • It selects rows from the two tables that have equal values in all matched columns. • If the columns having the same names have different data types, an error is returned. **Example:**

SELECT department\_id, department\_name,

location\_id, city

FROM departments

NATURAL JOIN locations ;

LOCATIONS table is joined to the DEPARTMENT table by the LOCATION\_ID column, which is the only column of the same name in both tables. If other common columns were present, the join would have used them all.

**Example:**

SELECT department\_id, department\_name,

location\_id, city

FROM departments

NATURAL JOIN locations

WHERE department\_id IN (20, 50);

**Creating Joins with the USING Clause**

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

• Use the USING clause to match only one column when more than one column matches. • Do not use a table name or alias in the referenced columns.

• The NATURAL JOIN and USING clauses are mutually exclusive.

**Example:**

SELECT l.city, d.department\_name

FROM locations l JOIN departments d USING (location\_id)

WHERE location\_id = 1400;

EXAMPLE:

SELECT e.employee\_id, e.last\_name, d.location\_id

FROM employees e JOIN departments d

USING (department\_id) ;

**Creating Joins with the ON Clause**

• The join condition for the natural join is basically an equijoin of all columns with the same name. • To specify arbitrary conditions or specify columns to join, the ON clause is used. • The join condition is separated from other searchconditions.

• The ON clause makes code easy to understand.

**Example:**

SELECT e.employee\_id, e.last\_name, e.department\_id,

d.department\_id, d.location\_id

FROM employees e JOIN departments d

ON (e.department\_id = d.department\_id);

EXAMPLE:

SELECT e.last\_name emp, m.last\_name mgr

FROM employees e JOIN employees m

ON (e.manager\_id = m.employee\_id);

INNER Versus OUTER Joins

• A join between two tables that returns the results of the inner join as well as unmatched rows left (or

right) tables is a left (or right) outer join.

• A join between two tables that returns the results of an inner join as well as the results of a left and right join is a full outer join.

**LEFT OUTER JOIN**

**Example:**

**SELECT e.last\_name, e.department\_id, d.department\_name**

FROM employees e

LEFT OUTER JOIN departments d

ON (e.department\_id = d.department\_id) ;

Example of LEFT OUTER JOIN

This query retrieves all rows in the EMPLOYEES table, which is the left table even if there is no match in the DEPARTMENTS table.

This query was completed in earlier releases as follows:

SELECT e.last\_name, e.department\_id, d.department\_name

FROM employees e, departments d

WHERE d.department\_id (+) = e.department\_id;

**RIGHT OUTER JOIN**

**Example:**

SELECT e.last\_name, e.department\_id, d.department\_name

FROM employees e

RIGHT OUTER JOIN departments d

ON (e.department\_id = d.department\_id) ;

This query retrieves all rows in the DEPARTMENTS table, which is the right table even if there is no

match in the EMPLOYEES table.

This query was completed in earlier releases as follows:

SELECT e.last\_name, e.department\_id, d.department\_name

FROM employees e, departments d

WHERE d.department\_id = e.department\_id (+);

**FULL OUTER JOIN**

**Example:**

SELECT e.last\_name, e.department\_id, d.department\_name

FROM employees e

FULL OUTER JOIN departments d

ON (e.department\_id = d.department\_id) ;

This query retrieves all rows in the EMPLOYEES table, even if there is no match in the DEPARTMENTS table. It alslso retrieves all rows in the DEPARTMENTS table, even if there is no match in the EMPLOYEES table.

**Find the Solution for the following:**

1. Write a query to display the last name, department number, and department name for all employees.

2. Create a unique listing of all jobs that are in department 80. Include the location of the department in the output.

3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission

8. Display the employee last name and department name for all employees who have an a(lowercase) in their last names. P

5. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

6. Display the employee last name and employee number along with their manager’s last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, Respectively

7. Modify lab4\_6.sql to display all employees including King, who has no manager. Order the results by the employee number.

8. Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label

9. Show the structure of the JOB\_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees

10. Create a query to display the name and hire date of any employee hired after employee Davies.

11. Display the names and hire dates for all employees who were hired before their managers, along with their manager’s names and hire dates. Label the columns Employee, Emp Hired, Manager, and Mgr Hired, respectively.

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Query(5) |  |
| Execution (5) |  |
| Viva(5) |  |
| Total (15) |  |
| Faculty Signature |  |

**EXERCISE-8**

**Aggregating Data Using Group Functions**

***Objectives***

After the completion of this exercise, the students be will be able to do the following:

• Identify the available group functions

• Describe the use of group functions

• Group data by using the GROUP BY clause

• Include or exclude grouped rows by using the HAVING clause

**What Are Group Functions?**

Group functions operate on sets of rows to give one result per group

**Types of Group Functions**

• AVG

• COUNT

• MAX

• MIN

• STDDEV

• SUM

• VARIANCE

Each of the functions accepts an argument. The following table identifies the options that you can use in the syntax:

**Group Functions: Syntax**

SELECT [*column*,] *group\_function(column), ...*

FROM *table*

[WHERE *condition*]

[GROUP BY *column*]

[ORDER BY *column*];

**Guidelines for Using Group Functions**

• DISTINCT makes the function consider only nonduplicate values; ALL makes it consider every value, including duplicates. The default is ALL and therefore does not need to be specified.

• The data types for the functions with an expr argument may be CHAR, VARCHAR2, NUMBER, or DATE.

• All group functions ignore null values.

**Using the AVG and SUM Functions**

You can use AVG and SUM for numeric data.

SELECT AVG(salary), MAX(salary),

MIN(salary), SUM(salary)

FROM employees

WHERE job\_id LIKE '%REP%';

**Using the MIN and MAX Functions**

You can use MIN and MAX for numeric, character, and date data types.

SELECT MIN(hire\_date), MAX(hire\_date)

FROM employees;

You can use the MAX and MIN functions for numeric, character, and date data types. example displays the most junior and most senior employees.

The following example displays the employee last name that is first and the employee last name that is last in an alphabetized list of all employees:

SELECT MIN(last\_name), MAX(last\_name)

FROM employees;

**Note:** The AVG, SUM, VARIANCE, and STDDEV functions can be used only with numeric data types. MAX and MIN cannot be used with LOB or LONG data types.

**Using the COUNT Function**

COUNT(\*) returns the number of rows in a table:

SELECT COUNT(\*)

FROM employees

WHERE department\_id = 50;

COUNT(*expr*) returns the number of rows with nonnull

values for the *expr*:

SELECT COUNT(commission\_pct)

FROM employees

WHERE department\_id = 80;

**Using the DISTINCT Keyword**

• COUNT(DISTINCT expr) returns the number of

distinct non-null values of the *expr*.

• To display the number of distinct department

values in the EMPLOYEES table:

SELECT COUNT(DISTINCT department\_id) FROM employees;

Use the DISTINCT keyword to suppress the counting of any duplicate values in a column. **Group Functions and Null Values**

Group functions ignore null values in the column:

SELECT AVG(commission\_pct)

FROM employees;

The NVL function forces group functions to include null values:

SELECT AVG(NVL(commission\_pct, 0))

FROM employees;

**Creating Groups of Data**

To divide the table of information into smaller groups. This can be done by using the GROUP BY clause.

**GROUP BY Clause Syntax**

SELECT *column*, *group\_function(column)*

FROM *table*

[WHERE *condition*]

[GROUP BY *group\_by\_expression*]

[ORDER BY *column*];

**In the syntax:**

*group\_by\_expression* specifies columns whose values determine the basis for grouping rows

**Guidelines**

• If you include a group function in a SELECT clause, you cannot select individual results as well, *unless* the individual column appears in the GROUP BY clause. You receive an error message if you fail to include the column list in the GROUP BY clause.

• Using a WHERE clause, you can exclude rows before dividing them into groups. • You must include the *columns* in the GROUP BY clause.

• You cannot use a column alias in the GROUP BY clause.

**Using the GROUP BY Clause**

All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

SELECT department\_id, AVG(salary)

FROM employees

GROUP BY department\_id ;

The GROUP BY column does not have to be in the SELECT list.

SELECT AVG(salary) FROM employees GROUP BY department\_id ;

You can use the group function in the ORDER BY clause:

SELECT department\_id, AVG(salary) FROM employees GROUP BY department\_id ORDER BY AVG(salary);

**Grouping by More Than One Column**

SELECT department\_id dept\_id, job\_id, SUM(salary) FROM employees

GROUP BY department\_id, job\_id ;

**Illegal Queries Using Group Functions**

Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP

**BY clause:**

SELECT department\_id, COUNT(last\_name) FROM employees;

You can correct the error by adding the GROUP BY clause:

SELECT department\_id, count(last\_name) FROM employees GROUP BY department\_id; You cannot use the WHERE clause to restrict groups.

• You use the HAVING clause to restrict groups.

• You cannot use group functions in the WHERE clause.

SELECT department\_id, AVG(salary) FROM employees WHERE AVG(salary) > 8000 GROUP BY department\_id;

You can correct the error in the example by using the HAVING clause to restrict groups:

SELECT department\_id, AVG(salary) FROM employees

HAVING AVG(salary) > 8000 GROUP BY department\_id;

**Restricting Group Results**

With the HAVING Clause .When you use the HAVING clause, the Oracle server restricts groups as follows:

1. Rows are grouped.

2. The group function is applied.

3. Groups matching the HAVING clause are displayed.

**Using the HAVING Clause**

SELECT department\_id, MAX(salary) FROM employees

GROUP BY department\_idHAVING MAX(salary)>10000 ;

The following example displays the department numbers and average salaries for those departments with a maximum salary that is greater than $10,000:

SELECT department\_id, AVG(salary) FROM employees GROUP BY department\_id HAVING max(salary)>10000;

Example displays the job ID and total monthly salary for each job that has a total payroll exceeding $13,000. The example excludes sales representatives and sorts the list by the total monthly salary.

SELECT job\_id, SUM(salary) PAYROLL FROM employees WHERE job\_id NOT LIKE '%REP%'

GROUP BY job\_id HAVING SUM(salary) > 13000 ORDER BY SUM(salary); **Nesting Group Functions**

**Display the maximum average salary:**

Group functions can be nested to a depth of two. The slide example displays the maximum average salary.

SELECT MAX(AVG(salary)) FROM employees GROUP BY department\_id; **Summary**

In this exercise, students should have learned how to:

• Use the group functions COUNT, MAX, MIN, and AVG

• Write queries that use the GROUP BY clause

• Write queries that use the HAVING clause

SELECT *column*, *group\_function*

FROM *table*

[WHERE *condition*]

[GROUP BY *group\_by\_expression*]

[HAVING *group\_condition*]

[ORDER BY *column*];

**Find the Solution for the following:**

Determine the validity of the following three statements. Circle either True or False.

1. Group functions work across many rows to produce one result per group. True/False

2. Group functions include nulls in calculations.

True/False

3. The WHERE clause restricts rows prior to inclusion in a group calculation. True/False

**The HR department needs the following reports:**

4. Find the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number

5. Modify the above query to display the minimum, maximum, sum, and

average salary for each job type.

6.Write a query to display the number of people with the same job. Generalize the query so that the user in the HR department is prompted for a job title.

7. Determine the number of managers without listing them. Label the column Number of Managers. *Hint: Use the MANAGER\_ID column to determine the number of managers.*

8. Find the difference between the highest and lowest salaries. Label the column DIFFERENCE.

9. Create a report to display the manager number and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is $6,000 or less. Sort the output in descending order of salary.

10. Create a query to display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

11. Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

12.Write a query to display each department’s name, location, number of employees, and the average salary for all the employees in that department. Label the column name-Location, Number of people, and salary respectively. Round the average salary to two decimal places.

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Query(5) |  |
| Execution (5) |  |
| Viva(5) |  |
| Total (15) |  |
| Faculty Signature |  |

**Practice Questions**

**Date Functions**

1. For DJs on Demand, display the number of months between the event\_date of the Vigil wedding and today’s date. Round to the nearest month.

2. Display the days between the start of last summer’s school vacation break and the day school started this year. Assume 30.5 days per month. Name the output “Days.”

3. Display the days between January 1 and December 31.

4. Using one statement, round today's date to the nearest month and nearest year and truncate it to the nearest month and nearest year. Use an alias for each column.

5. What is the last day of the month for June 2005? Use an alias for the output. 6.Display the number of years between the Global Fast Foods employee Bob Miller’s birthday and

today. Round to the nearest year.

7.Your next appointment with the dentist is six months from today. On what day will you go to the dentist? Name the output, “Appointment.”

8.The teacher said you have until the last day of this month to turn in your research paper. What day will this be? Name the output, “Deadline.”

9.How many months between your birthday this year and January 1 next year? 10.What’s the date of the next Friday after your birthday this year? Name the output, “First Friday.”

11. Name a date function that will return a number.

12.Name a date function that will return a date.

13.Give one example of why it is important for businesses to be able to manipulate date data?

**Conversion Functions**

In each of the following exercises, feel free to use labels for the converted column to make the output more readable.

1. List the last names and birthdays of Global Fast Food Employees. Convert the birth dates to character data in the Month DD, YYYY format. Suppress any leading zeros.

2. Convert January 3, 04, to the default date format 03-Jan-2004.

3. Format a query from the Global Fast Foods f\_promotional\_menus table to print out the start\_date of promotional code 110 as: The promotion began on the tenth of February 2004.

4. Convert today’s date to a format such as: “Today is the Twentieth of March, Two Thousand Four”

5. List the ID, name and salary for all Global Fast Foods employees. Display salary with a $ sign and two decimal places.

| Evaluation Procedure | Marks awarded |
| --- | --- |
| Practice Evaluation (5) |  |
| Viva(5) |  |
| Total (10) |  |
| Faculty Signature |  |

**EXERCISE-9**

**Sub queries**

**Objectives**

After completing this lesson, you should be able to do the following:

• Define subqueries

• Describe the types of problems that subqueries can solve

• List the types of subqueries

• Write single-row and multiple-row subqueries

**Using a Subquery to Solve a Problem**

Who has a salary greater than Abel’s?

**Main query:**

Which employees have salaries greater than Abel’s salary?

**Subquery:**

What is Abel’s salary?

**Subquery Syntax**

SELECT *select\_list* FROM *table* WHERE *expr operator* (SELECT *select\_list* FROM *table*); • The subquery (inner query) executes once before the main query (outer query). • The result of the subquery is used by the main query.

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, including the following:

• WHERE clause

• HAVING clause

• FROM clause

**In the syntax:**

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

**Note:** Comparison conditions fall into two classes: single-row operators

(>, =, >=, <, <>, <=) and multiple-row operators (IN, ANY, ALL). statement. The subquery generally executes first, and its output is used to complete the query condition for the main (or outer) query

**Using a Subquery**

SELECT last\_name FROM employees WHERE salary > (SELECT salary FROM employees WHERE last\_name = 'Abel');

The inner query determines the salary of employee Abel. The outer query takes the result of the inner query and uses this result to display all the employees who earn more than this amount.

**Guidelines for Using Subqueries**

• Enclose subqueries in parentheses.

• Place subqueries on the right side of the comparison condition.

**•** The ORDER BY clause in the subquery is not needed unless you are performing Top-N analysis. • Use single-row operators with single-row

subqueries, and use multiple-row operators with multiple-row subqueries.

**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.

**Single-Row Subqueries**

• Return only one row

• Use single-row comparison operators

**Example**

Display the employees whose job ID is the same as that of employee 141:

SELECT last\_name, job\_id FROM employees WHERE job\_id = (SELECT job\_id FROM employees

WHERE employee\_id = 141);

Displays employees whose job ID is the same as that of employee 141 and whose salary is greater than that of employee 143.

SELECT last\_name, job\_id, salary FROM employeesWHERE job\_id =(SELECT job\_id FROM employees WHERE employee\_id = 141) AND salary > (SELECT salary FROM employees WHERE employee\_id = 143);

**Using Group Functions in a Subquery**

Displays the employee last name, job ID, and salary of all employees whose salary is equal to the minimum salary. The MIN group function returns a single value (2500) to the outer query.

SELECT last\_name, job\_id, salary FROM employees WHERE salary = (SELECT MIN(salary) FROM employees);

**The HAVING Clause with Subqueries**

• The Oracle server executes subqueries first.

• The Oracle server returns results into the HAVING clause of the main query. Displays all the departments that have a minimum salary greater than that of department 50.

SELECT department\_id, MIN(salary)

FROM employees

GROUP BY department\_id

HAVING MIN(salary) >

(SELECT MIN(salary)

FROM employees

WHERE department\_id = 50);

**Example**

**Find the job with the lowest average salary.**

SELECT job\_id, AVG(salary)

FROM employees

GROUP BY job\_id

HAVING AVG(salary) = (SELECT MIN(AVG(salary))

FROM employees

GROUP BY job\_id);

**What Is Wrong in this Statements?**

SELECT employee\_id, last\_name

FROM employees

WHERE salary =(SELECT MIN(salary) FROM employees GROUP BY department\_id); Will This Statement Return Rows?

SELECT last\_name, job\_id

FROM employees

WHERE job\_id =(SELECT job\_id FROM employees WHERE last\_name = 'Haas'); **Multiple-Row Subqueries**

• Return more than one row

• Use multiple-row comparison operators

**Example**

Find the employees who earn the same salary as the minimum salary for each department.

SELECT last\_name, salary, department\_id FROM employees WHERE salary IN (SELECT MIN(salary)

FROM employees GROUP BY department\_id);

Using the ANY Operator in Multiple-Row Subqueries

SELECT employee\_id, last\_name, job\_id, salary FROM employees WHERE salary < ANY (SELECT salary FROM employees WHERE job\_id = 'IT\_PROG') AND job\_id <> 'IT\_PROG';

Displays employees who are not IT programmers and whose salary is less than that of any IT programmer. The maximum salary that a programmer earns is $9,000.

< ANY means less than the maximum. >ANY means more than the minimum. =ANY is equivalent to IN.

**Using the ALL Operator in Multiple-Row Subqueries**

SELECT employee\_id, last\_name, job\_id, salary

FROM employees

WHERE salary < ALL (SELECT salary FROM employees WHERE job\_id = 'IT\_PROG') AND job\_id <> 'IT\_PROG';

Displays employees whose salary is less than the salary of all employees with a job ID of IT\_PROG and whose job is not IT\_PROG.

⮚ ALL means more than the maximum, and <ALL means less than the minimum. The NOT operator can be used with IN, ANY, and ALL operators.

**Null Values in a Subquery**

SELECT emp.last\_name FROM employees emp

WHERE emp.employee\_id NOT IN (SELECT mgr.manager\_id FROM employees mgr);

Notice that the null value as part of the results set of a subquery is not a problem if you use the IN operator. The IN operator is equivalent to =ANY. For example, to display the employees who have subordinates, use the following SQL statement:

SELECT emp.last\_name

FROM employees emp

WHERE emp.employee\_id IN (SELECT mgr.manager\_id FROM employees mgr); Display all employees who do not have any subordinates:

SELECT last\_name FROM employees

WHERE employee\_id NOT IN (SELECT manager\_id FROM employees WHERE manager\_id IS NOT NULL);

**Find the Solution for the following:**

1. The HR department needs a query that prompts the user for an employee last name. The query then displays the last name and hire date of any employee in the same department as the employee whose name they supply (excluding that employee). For example, if the user enters Zlotkey, find all employees who work with Zlotkey (excluding Zlotkey).

2. Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in order of ascending salary.