RAJALAKSHMI ENGINEERING COLLEGE RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

CS19443

DATABASE MANAGEMENT SYSTEMS LAB

Laboratory Record Note Book

Name : **SAKTHI M S**

Year / Branch / Section : **II / B.E CSE / D**

University Register No. : **2116220701240**

College Roll No. : **220701240**

Semester : **IV**

Academic Year : **2023-2024**

INDEX

Name : _____ Branch : _____ Sec : _____ Roll No : _____

S.No.	Date	Title	Page No.	Teacher's Sign / Remarks

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*;
Example:

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));
```

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));
```

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));
```

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), FIRST_NAME VARCHAR2(25), LAST_NAME VARCHAR2(25), SALARY
NUMBER(9), DEPT_ID NUMBER(7));
```

5. Drop the EMP table.

```
DROP TABLE EMP;
```

6. Rename the EMPLOYEES2 table as EMP.

```
RENAME EMPLOYEES2 TO EMP;
```

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

```
COMMENT ON TABLE DEPT is 'This is a table';
DESC emp;
```

8. Drop the First_name column from the EMP table and confirm it.

```
ALTER TABLE emp
DROP COLUMN first_name;
```

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)

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

3. Display the table with values.

```
SELECT * FROM MY_EMPLOYEE;
```

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 (ID, Last_name, First_name, Userid, Salary) VALUES (3, 'Biri', 'Ben', 'bbiri', 1100);
INSERT INTO MY_EMPLOYEE (ID, Last_name, First_name, Userid, Salary) VALUES (4, 'Newman', 'Chad',
'Cnewman', 750);
```

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;
```

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;
```

8. Delete Betty dancs from MY _EMPLOYEE table.

```
DELETE FROM MY_EMPLOYEE
WHERE First_name = 'Betty' AND Last_name = 'Dancs';
```

9. Empty the fourth row of the emp table.

```
DELETE FROM MY_EMPLOYEE
WHERE ID = 4;
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	

Viva(5)	
Total (15)	
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:

- Setting null value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A null value is not equivalent to a value of zero.
- A null value will always evaluate to null in any expression.
- When a column name is defined as not null, that column becomes a mandatory i.e., the user has to enter data into it.
- 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.

ALTER TABLE EMP
ADD CONSTRAINT my_emp_id_pk PRIMARY KEY (ID);

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.

ALTER TABLE DEPT
ADD CONSTRAINT my_dept_id_pk PRIMARY KEY (ID);
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.

ALTER TABLE EMP
ADD DEPT_ID NUMBER;

ALTER TABLE EMP
ADD CONSTRAINT my_emp_dept_id_fk FOREIGN KEY (DEPT_ID)
REFERENCES DEPT(ID);

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.

ALTER TABLE EMP
ADD COMMISSION NUMBER(5, 2);
ALTER TABLE EMP
ADD CONSTRAINT chk_commission_positive CHECK (COMMISSION > 0);

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
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

SELECT employee_id, last_name, sal * 12 AS "ANNUAL SALARY"
FROM employees;

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

DESC DEPARTMENTS;

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

SELECT employee_id AS "EMPLOYEE NUMBER", last_name, job_id AS "JOB CODE", hire_date AS STARTDATE FROM employees;

4. Provide an alias STARTDATE for the hire date.

SELECT employee_id AS "EMPLOYEE NUMBER", last_name, job_id AS "JOB CODE", hire_date AS STARTDATE
FROM employees;

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

SELECT DISTINCT job_id
FROM employees;

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

SELECT last_name || ', ' || job_id AS "EMPLOYEE and TITLE"
FROM employees;

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

SELECT *
FROM employees;

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
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.

```
SELECT last_name, salary
FROM employees
WHERE salary > 12000;
```

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

```
SELECT last_name, department_id
FROM employees
WHERE employee_id = 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)

```
SELECT last_name, salary
FROM employees
WHERE salary NOT BETWEEN 5000 AND 12000;
```

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)

```
SELECT last_name, job_id, hire_date
FROM employees
WHERE hire_date BETWEEN TO_DATE('1998-02-20', 'YYYY-MM-DD') AND
TO_DATE('1998-05-01', 'YYYY-MM-DD')
ORDER BY hire_date ASC;
```

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

```
SELECT last_name, department_id
FROM employees
WHERE department_id IN (20, 50)
ORDER BY last_name;
```

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)

```
SELECT last_name AS EMPLOYEE, salary AS "MONTHLY SALARY"
FROM employees
WHERE department_id IN (20, 50)
 AND salary BETWEEN 5000 AND 12000
ORDER BY last_name;
```

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

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date LIKE '1994%';
```

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

```
SELECT last_name, job_title
FROM employees
WHERE manager_id 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)

```
SELECT last_name, salary, commission_pct AS commission
FROM employees
WHERE commission_pct IS NOT NULL
ORDER BY salary DESC, commission_pct DESC;
```

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

```
SELECT last_name
FROM employees
WHERE SUBSTR(last_name, 3, 1) = 'a';
```

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

```
SELECT last_name
FROM employees
WHERE last_name LIKE '%a%' AND last_name LIKE '%e%';
```

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)

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id IN ('SA_REP', 'ST_CLERK')
```

AND salary NOT IN (2500, 3500, 7000);

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

```
SELECT last_name, salary, commission_pct AS commission
FROM employees
WHERE commission_pct = 0.20;
```

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

EXERCISE-6

Single Row Functions

Objective

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

SINGLE-ROW FUNCTIONS

GENERAL

DATE

NUMBER

CONVERSION

● 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

Function	Result
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

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

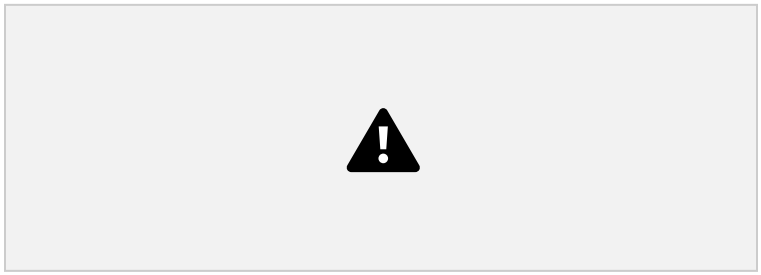
Display the employee number, hire date, number of months employed, sixmonth review date, first Friday after hire date, and last day of the hire month for all employees who have been employed for fewer than 70 months.

```
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

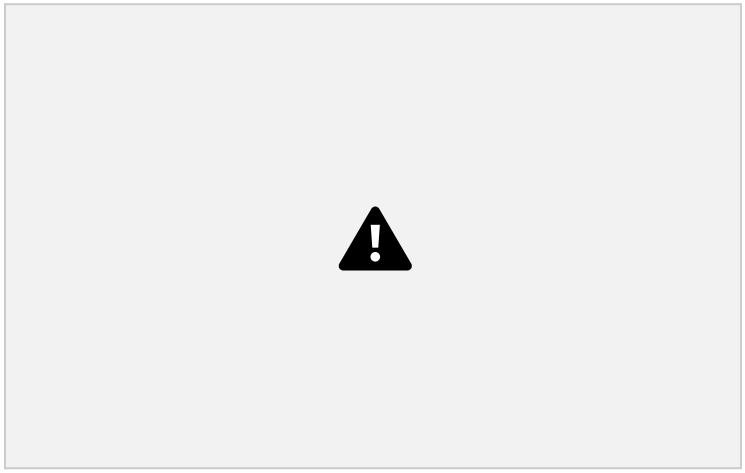
For assignments, the Oracle server can automatically convert the

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

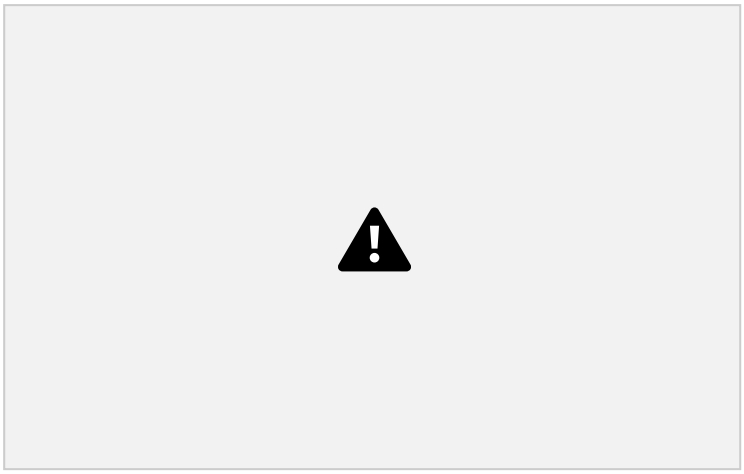
TO_CHAR(date, 'format_model')

The format model:

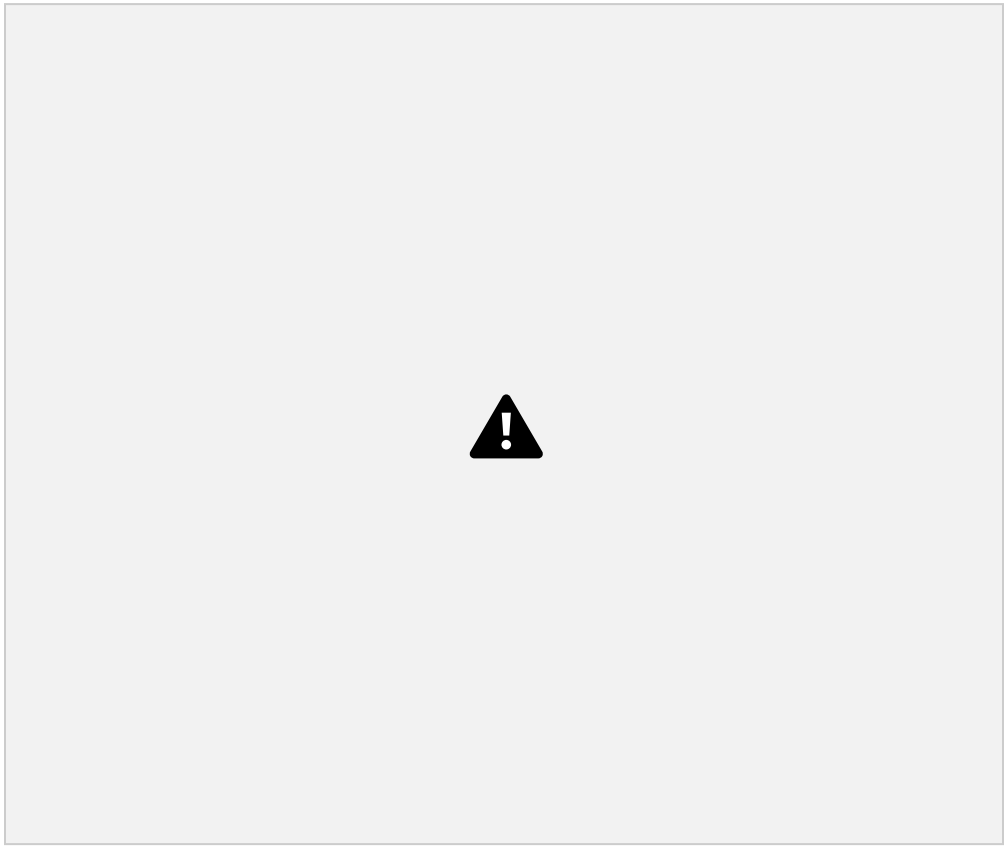
• Must be enclosed by single quotation marks
• Is case-sensitive
• Can include any valid date format element
• Has an fm element to remove padded blanks or suppress leading zeros
• Is separated from the date value by a comma

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM employees WHERE last_name = 'Higgins';
```

Elements of the Date Format Model

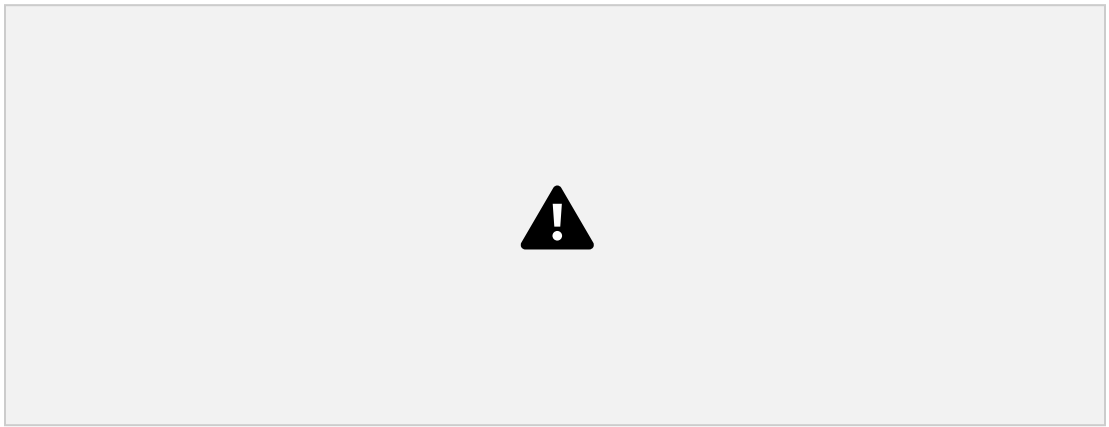


Sample Format Elements of Valid Date



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

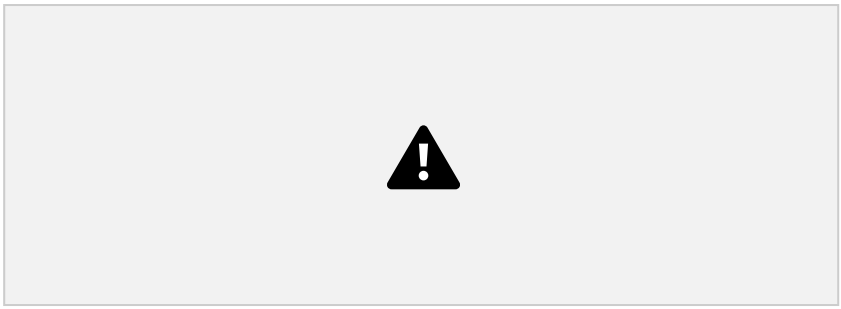
```
SELECT last_name,
TO_CHAR(hire_date, 'fmDD Month YYYY') AS HIREDATE
FROM employees;
```

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;

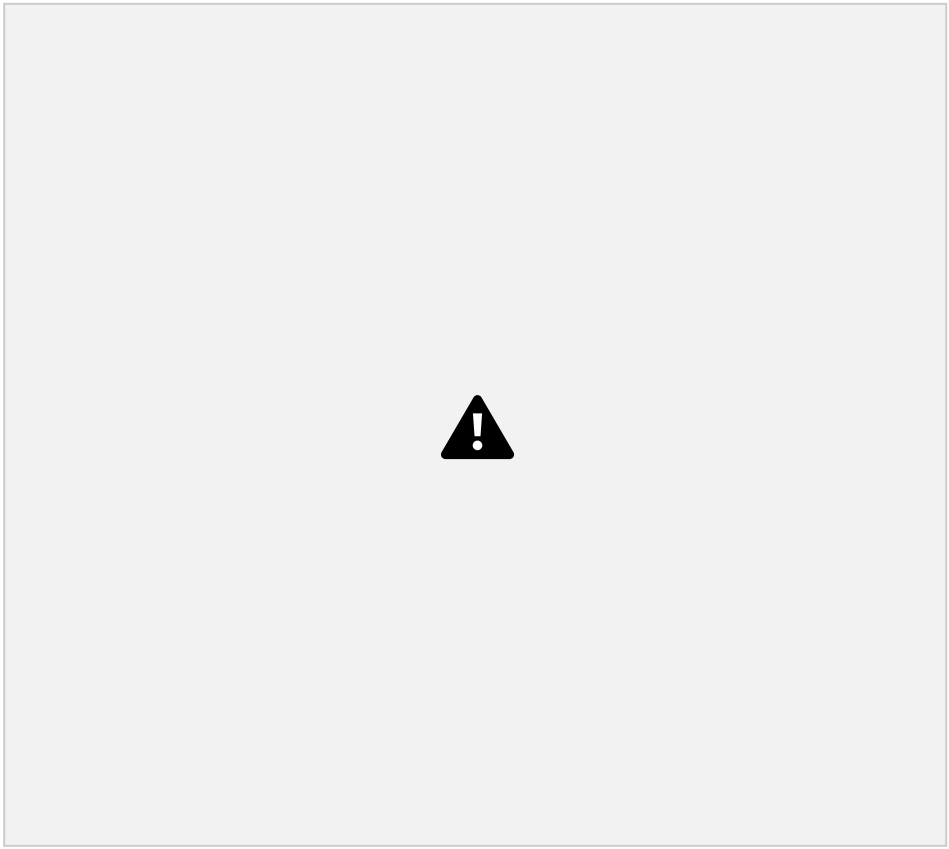
Using the TO_CHAR Function with Numbers

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.

```
SELECT CURRENT_DATE AS "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.

```
SELECT employee_id, last_name, salary,
 ROUND(salary * 1.155) AS "New Salary"
FROM employees;
```

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.

```
SELECT employee_id, last_name, salary,
 ROUND(salary * 1.155) AS "New Salary",
 ROUND(salary * 0.155) AS "Increase"
FROM employees;
```

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.

```
SELECT INITCAP(last_name) AS "Employee Name",
 LENGTH(last_name) AS "Name Length"
FROM employees
WHERE last_name LIKE 'J%' OR last_name LIKE 'A%' OR last_name LIKE
'M%' ORDER BY last_name;
```

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.

```
User_input = 'H'
SELECT last_name FROM employees
WHERE last_name LIKE User_input || '%';
```

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.

```
SELECT last_name,
 CEIL(MONTHS_BETWEEN(CURRENT_DATE, hire_date)) AS "MONTHS_WORKED"
FROM employees
ORDER BY "MONTHS_WORKED";
```

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.

```
SELECT last_name,
 salary AS "Current Salary",
 salary * 3 AS "Dream Salary"
FROM employees;
```

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.

```
SELECT last_name,
```

```
 LPAD('$' || TO_CHAR(salary, '99999.99'), 15, ' ') AS "SALARY"
FROM employees;
```

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."

```
SELECT last_name,
 hire_date,
 NEXT_DAY(ADD_MONTHS(hire_date, 6), 'MONDAY') AS "REVIEW"
FROM employees;
```

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.

```
SELECT last_name,
 hire_date,
 TO_CHAR(hire_date, 'DAY') AS "DAY"
FROM employees
ORDER BY TO_CHAR(hire_date, 'D');
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
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.

```
SELECT e.last_name, d.department_id, d.department_name
```

```
FROM employees e
JOIN departments d ON e.department_id = d.department_id;
```

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

```
SELECT DISTINCT j.job_title, l.city, l.state_province
FROM jobs j
JOIN employees e ON j.job_id = e.job_id
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id
WHERE d.department_id = 80;
```

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

```
SELECT e.last_name, d.department_name, l.location_id, l.city
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id
WHERE e.commission_pct IS NOT NULL;
```

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

```
SELECT e.last_name, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id
WHERE e.last_name LIKE '%a%';
```

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

```
SELECT e.last_name, j.job_title, e.department_id, d.department_name
FROM employees e
JOIN jobs j ON e.job_id = j.job_id
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id
WHERE l.city = '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

```
SELECT e.last_name AS "Employee", e.employee_id AS "Emp#",
 m.last_name AS "Manager", m.employee_id AS "Mgr#"
FROM employees e
LEFT JOIN employees m ON e.manager_id = m.employee_id;
```

7. Modify lab4_6.sql to display all employees including King, who has no manager. Order the results by the employee number.

```
SELECT e.last_name, e.employee_id, e.manager_id
FROM employees e
UNION
SELECT 'King', 100, NULL
FROM dual
ORDER BY e.employee_id;
```

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

```
SELECT e1.last_name, e1.department_id, e2.last_name AS "Colleague"
FROM employees e1
JOIN employees e2 ON e1.department_id = e2.department_id
WHERE e1.employee_id = 101;
```

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

```
CREATE TABLE job_grades (
 grade CHAR(1),
 lowest_sal NUMBER(8, 2) NOT NULL,
 highest_sal NUMBER(8, 2) NOT NULL
);
```

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

```
SELECT e.last_name, e.hire_date
FROM employees e
WHERE e.hire_date > (SELECT hire_date FROM employees WHERE last_name = '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.

```
SELECT e.last_name AS "Employee", e.hire_date AS "Emp Hired",
 m.last_name AS "Manager", m.hire_date AS "Mgr Hired"
FROM employees e
JOIN employees m ON e.manager_id = m.employee_id
WHERE e.hire_date < m.hire_date;
```

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?

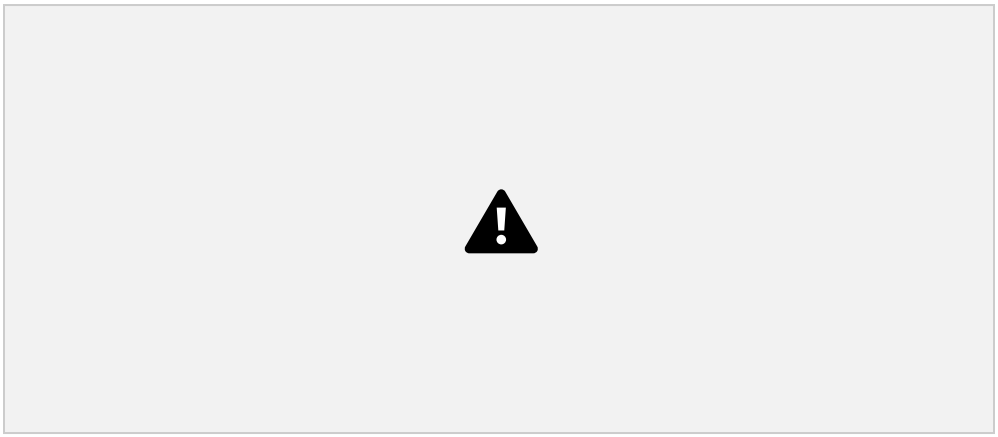
Group functions operate on sets of rows to give one result per group

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 -- True

2. Group functions include nulls in calculations.
True/False -- False

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

```
SELECT
 ROUND(MAX(salary)) AS Maximum,
 ROUND(MIN(salary)) AS Minimum,
 ROUND(SUM(salary)) AS Sum,
 ROUND(AVG(salary)) AS Average
FROM employees;
```

5. Modify the above query to display the minimum, maximum, sum, and average salary for each job type.

```
SELECT job_id,
 ROUND(MIN(salary)) AS Minimum,
 ROUND(MAX(salary)) AS Maximum,
 ROUND(SUM(salary)) AS Sum,
 ROUND(AVG(salary)) AS Average
FROM employees
GROUP BY job_id;
```

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.

```
SELECT job_id, COUNT(*) AS "Number of People"
FROM employees
WHERE job_id = :user_input_job_id
GROUP BY job_id;
```

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.*

```
SELECT COUNT(DISTINCT manager_id) AS "Number of Managers"
FROM employees;
```

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

```
SELECT ROUND(MAX(salary) - MIN(salary)) AS Difference
FROM employees;
```

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.

```
SELECT manager_id, MIN(salary) AS "Lowest Salary"
FROM employees
WHERE manager_id IS NOT NULL
GROUP BY manager_id
HAVING MIN(salary) > 6000
ORDER BY "Lowest Salary" DESC;
```

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.

```
SELECT
 COUNT(*) AS "Total Employees",
 SUM(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1995 THEN 1 ELSE 0 END) AS
"Hired in 1995",
 SUM(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1996 THEN 1 ELSE 0 END) AS
"Hired in 1996",
 SUM(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1997 THEN 1 ELSE 0 END) AS
"Hired in 1997",
 SUM(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1998 THEN 1 ELSE 0 END) AS
"Hired in 1998"
FROM employees;
```

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.

```
SELECT
```

```
 j.job_title AS "Job",
 ROUND(AVG(CASE WHEN d.department_id = 20 THEN e.salary ELSE NULL END), 2) AS
"Salary in Dept 20",
 ROUND(AVG(CASE WHEN d.department_id = 50 THEN e.salary ELSE NULL END), 2) AS
"Salary in Dept 50",
 ROUND(AVG(CASE WHEN d.department_id = 80 THEN e.salary ELSE NULL END), 2) AS
"Salary in Dept 80",
 ROUND(AVG(CASE WHEN d.department_id = 90 THEN e.salary ELSE NULL END), 2) AS
"Salary in Dept 90",
 ROUND(SUM(e.salary), 2) AS "Total Salary"
FROM employees e
JOIN jobs j ON e.job_id = j.job_id
JOIN departments d ON e.department_id = d.department_id
WHERE d.department_id IN (20, 50, 80, 90)
GROUP BY j.job_title;
```

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.

```
SELECT
 d.department_name AS "Department Name",
 l.city || ', ' || l.state_province AS "Location",
 COUNT(*) AS "Number of People",
 ROUND(AVG(e.salary), 2) AS "Salary"
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id
GROUP BY d.department_name, l.city, l.state_province;
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
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).

SELECT last_name, hire_date
FROM employees
WHERE department_id = (SELECT department_id FROM employees WHERE last_name = '&name')
AND last_name <> '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.
SELECT employee_id, last_name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);

3. Write a query that displays the employee number and last name of all employees who work in a

department with any employee whose last name contains a *u*.

```
SELECT employee_id, last_name
FROM employees
WHERE department_id IN
 (SELECT department_id FROM employees WHERE last_name LIKE '%u%');
```

4. The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.

```
SELECT last_name, department_id, job_id
FROM employees
WHERE department_id IN
 (SELECT department_id FROM departments WHERE location_id = 1700);
```

5. Create a report for HR that displays the last name and salary of every employee who reports to King.

```
SELECT last_name, salary
FROM employees
WHERE manager_id = (SELECT employee_id FROM employees WHERE last_name = 'King');
```

6. Create a report for HR that displays the department number, last name, and job ID for every employee in the Executive department.

```
SELECT department_id, last_name, job_id
FROM employees
WHERE department_id = (SELECT department_id FROM departments WHERE
department_name = 'Executive');
```

7. Modify the query 3 to display the employee number, last name, and salary of all employees who earn more than the average salary and who work in a department with any employee whose last name contains a *u*.

```
SELECT e.employee_id, e.last_name, e.salary
FROM employees e
WHERE e.department_id IN
 (SELECT d.department_id FROM departments d WHERE d.location_id = 1700)
AND e.salary > (SELECT AVG(salary) FROM employees);
```

Evaluation Procedure	Marks awarded
Query(5)	

Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

EXERCISE-10

USING THE SET OPERATORS

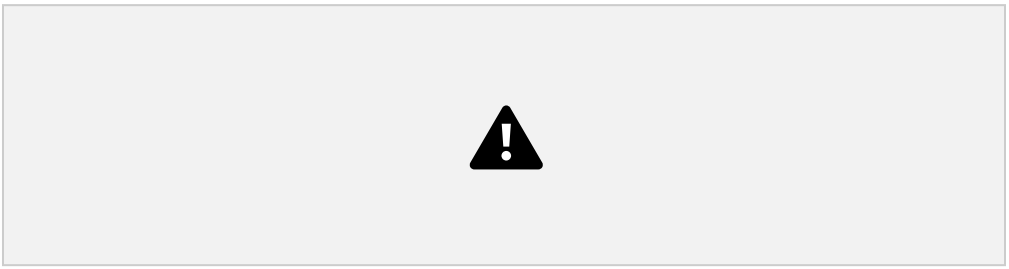
Objectives

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

- Describe set operators
- Use a set operator to combine multiple queries into a single query
- Control the order of rows returned

The set operators combine the results of two or more component queries into one result.

Queries containing set operators are called *compound queries*.



The tables used in this lesson are:

- EMPLOYEES: Provides details regarding all current employees

- JOB_HISTORY: Records the details of the start date and end date of the former job, and the job identification number and department when an employee switches jobs

UNION Operator

Guidelines

- The number of columns and the data types of the columns being selected must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.

- UNION operates over all of the columns being selected.

- NULL values are not ignored during duplicate checking.

- The IN operator has a higher precedence than the UNION operator.

• By default, the output is sorted in ascending order of the first column of the SELECT

clause. **Example:**

Display the current and previous job details of all employees. Display each employee only once.

```
SELECT employee_id, job_id FROM employees UNION SELECT employee_id,
job_id FROM job_history;
```

Example:

```
SELECT employee_id, job_id, department_id
FROM employees
UNION
SELECT employee_id, job_id, department_id
FROM job_history;
```

UNION ALL Operator

Guidelines

The guidelines for UNION and UNION ALL are the same, with the following two exceptions that pertain to UNION ALL:
• Unlike UNION, duplicate rows are not eliminated and the output is not sorted by default. • The DISTINCT keyword cannot be used.

Example:

Display the current and previous departments of all employees.

```
SELECT employee_id, job_id, department_id
FROM employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM job_history
ORDER BY employee_id;
```

INTERSECT Operator

Guidelines

• The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
• Reversing the order of the intersected tables does not alter the result.
• INTERSECT does not ignore NULL values.

Example:

Display the employee IDs and job IDs of those employees who currently have a job title that is the same as their job title when they were initially hired (that is, they changed jobs but have now gone back to doing their original job).

```
SELECT employee_id, job_id FROM employees
INTERSECT
```

```
SELECT employee_id, job_id
FROM job_history;
```

Example

```
SELECT employee_id, job_id, department_id
FROM employees
INTERSECT
SELECT employee_id, job_id, department_id
FROM job_history;
```

MINUS Operator

Guidelines

• The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.

• All of the columns in the WHERE clause must be in the SELECT clause for the MINUS operator to work.

Example:

Display the employee IDs of those employees who have not changed their jobs even once.

```
SELECT employee_id,job_id
FROM employees
MINUS
SELECT employee_id,job_id
FROM job_history;
```

Find the Solution for the following:

1. The HR department needs a list of department IDs for departments that do not contain the job ID ST_CLERK. Use set operators to create this report.

```
SELECT DISTINCT department_id
FROM employees
WHERE department_id NOT IN (SELECT department_id FROM employees WHERE job_id =
'ST_CLERK');
```

2. The HR department needs a list of countries that have no departments located in them. Display the country ID and the name of the countries. Use set operators to create this report.

```
SELECT country_id, country_name
FROM countries
WHERE country_id NOT IN
 (SELECT DISTINCT location_id FROM departments);
```

3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID using set operators.

```
SELECT job_id, department_id
FROM employees
WHERE department_id = 10
UNION ALL
SELECT job_id, department_id
```

```
FROM employees
WHERE department_id = 50
UNION ALL
SELECT job_id, department_id
FROM employees
WHERE department_id = 20;
```

4. Create a report that lists the employee IDs and job IDs of those employees who currently have a job title that is the same as their job title when they were initially hired by the company (that is, they changed jobs but have now gone back to doing their original job).

```
SELECT e.employee_id, e.job_id
FROM employees e
WHERE e.job_id IN
 (SELECT j.job_id FROM job_history j WHERE j.start_date = e.hire_date);
```

5. The HR department needs a report with the following specifications:

- Last name and department ID of all the employees from the EMPLOYEES table, regardless of whether or not they belong to a department.

- Department ID and department name of all the departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them Write a compound query to accomplish this.

```
SELECT last_name, department_id FROM employees
UNION ALL
SELECT NULL, department_id, department_name FROM departments;
```

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

EXERCISE-11

CREATING VIEWS

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

● Describe a view
● Create, alter the definition of, and drop a view
● Retrieve data through a view
● Insert, update, and delete data through a view
● Create and use an inline view

View

A view is a logical table based on a table or another view. A view contains no data but is like a window through which data from tables can be viewed or changed.The tables on which a view is based are called base tables.

Advantages of Views

● To restrict data access
● To make complex queries easy
● To provide data independence
● To present different views of the same data

Classification of views

1. Simple view
2. Complex view

Feature	Simple	Complex
No. of tables	One	One or more
Contains functions	No	Yes
Contains groups of data	No	Yes
DML operations thr' view	Yes	Not always

Creating a view

Syntax
CREATE OR REPLACE FORCE/NOFORCE VIEW view_name AS Subquery WITH CHECK OPTION CONSTRAINT constraint WITH READ ONLY CONSTRAINT constraint;

FORCE - Creates the view regardless of whether or not the base tables exist.

NOFORCE - Creates the view only if the ase table exist.

WITH CHECK OPTION CONSTRAINT-specifies that only rows accessible to the view can be inserted or updated.

WITH READ ONLY CONSTRAINT-ensures that no DML operations can be performed on the view.
Example: 1 (Without using Column aliases)
Create a view EMPVU80 that contains details of employees in

department80. **Example 2:**

CREATE VIEW empvu80 AS SELECT employee_id, last_name, salary FROM employees
WHERE department_id=80;

Example:1 (Using column aliases)

CREATE VIEW salvu50
AS SELECT employee_id,id_number, last_name NAME, salary *12 ANN_SALARY
FROM employees
WHERE department_id=50;

Retrieving data from a view

Example:
SELECT * from salvu50;

Modifying a view

A view can be altered without dropping, re-creating.

Example: (Simple view)

Modify the EMPVU80 view by using CREATE OR REPLACE.

CREATE OR REPLACE VIEW empvu80 (id_number, name, sal, department_id)
AS SELECT employee_id,first_name, last_name, salary, department_id FROM
employees
WHERE department_id=80;

Example: **(complex view)**

CREATE VIEW dept_sum_vu (name, minsal, maxsal,avgsal)
AS SELECT d.department_name, MIN(e.salary), MAX(e.salary), AVG(e.salary)
FROM employees e, department d
WHERE e.deparment_id=d.deparment_id
GROUP BY d.department_name;

Rules for performing DML operations on view

● Can perform operations on simple views
● Cannot remove a row if the view contains the following:
▪ Group functions
▪ Group By clause
▪ Distinct keyword
● Cannot modify data in a view if it contains
▪ Group functions
▪ Group By clause
▪ Distinct keyword
▪ Columns contain by expressions
▪
● Cannot add data thr' a view if it contains
▪ Group functions
▪ Group By clause
▪ Distinct keyword
▪ Columns contain by expressions

▪ NOT NULL columns in the base table that are not selected by the view

Example: (Using the WITH CHECK OPTION clause)

```
CREATE OR REPLACE VIEW empvu20
 AS SELECT *
 FROM employees
 WHERE department_id=20
 WITH CHECK OPTION CONSTRAINT empvu20_ck;
```

Note: Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

Example – (Execute this and note the error)

```
UPDATE empvu20 SET department_id=10 WHERE employee_id=201;
```

Denying DML operations

Use of WITH READ ONLY option.
Any attempt to perform a DML on any row in the view results in an oracle server error.

Try this code:

```
CREATE OR REPLACE VIEW empvu10(employee_number, employee_name,job_title)
AS SELECT employee_id, last_name, job_id
FROM employees
WHERE department_id=10
WITH READ ONLY;
```

Find the Solution for the following:

1. Create a view called EMPLOYEE_VU based on the employee numbers, employee names and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.

```
CREATE OR REPLACE VIEW EMPLOYEE_VU AS
SELECT employee_id, last_name AS employee, department_id
FROM employees;
```

2. Display the contents of the EMPLOYEES_VU view.

```
SELECT * FROM EMPLOYEE_VU;
```

3. Select the view name and text from the USER_VIEWS data dictionary views.

```
SELECT view_name, text
FROM user_views;
```

4. Using your EMPLOYEES_VU view, enter a query to display all employees names and department.

```
SELECT employee, department_id
FROM EMPLOYEE_VU;
```

5. Create a view named DEPT50 that contains the employee number, employee last names and department numbers for all employees in department 50.Label the view columns EMPNO, EMPLOYEE and DEPTNO. Do not allow an employee to be reassigned to another department through the view.

```
CREATE OR REPLACE VIEW DEPT50 AS
SELECT employee_id AS EMPNO, last_name AS EMPLOYEE, department_id AS
DEPTNO FROM employees
WHERE department_id = 50;
```

6. Display the structure and contents of the DEPT50 view.

```
DESCRIBE DEPT50;
SELECT * FROM DEPT50;
```

7. Attempt to reassign Matos to department 80.

```
UPDATE DEPT50 SET DEPTNO = 80 WHERE EMPLOYEE = 'Matos';
```

8. Create a view called SALARY_VU based on the employee last names, department names, salaries, and salary grades for all employees. Use the Employees, DEPARTMENTS and JOB_GRADE tables. Label the column Employee, Department, salary, and Grade respectively.

```
CREATE OR REPLACE VIEW SALARY_VU AS
SELECT e.last_name AS Employee, d.department_name AS
Department,  e.salary, j.grade
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN job_grade j ON e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	

Viva(5)	
Total (15)	
Faculty Signature	

EXERCISE 12

Intro to Constraints; NOT NULL and UNIQUE Constraints

Global Fast Foods has been very successful this past year and has opened several new stores. They need to add a table to their database to store information about each of their store's locations. The owners want to make sure that all entries have an identification number, date opened, address, and city and that no other entry in the table can have the same email address. Based on this information, answer the following questions about the global_locations table. Use the table for your answers.

Global Fast Foods global_locations Table						
NAME	**TYPE**	**LENGTH**	**PRECISION**	**SCALE**	**NULLABLE**	**DEFAULT**
Id						
name						
date_opened						
address						
city						
zip/postal code						
phone						
email						
manager_id						
Emergency contact						

1. What is a "constraint" as it relates to data integrity?

A constraint in the context of data integrity refers to predefined rules or conditions that are enforced on a database to ensure the accuracy and consistency of data

2. What are the limitations of constraints that may be applied at the column level and at the table level?

• Constraints can be applied at both the **column level** and the **table level**. Here are the key differences:

- **Column-Level Constraints**:
  - Apply only to a specific column.
  - Examples include **NOT NULL**, **UNIQUE**, and **CHECK** constraints.
  - Cannot involve multiple columns.
- **Table-Level Constraints**:
  - Apply to the entire table.
  - Can involve multiple columns.
  - Examples include **PRIMARY KEY**, **FOREIGN KEY**, and **CHECK** constraints.

3. Why is it important to give meaningful names to constraints?

For clarity, maintenance, documentation and debugging.

4. Based on the information provided by the owners, choose a datatype for each column. Indicate the length, precision, and scale for each NUMBER datatype.

Id: NUMBER (10,0)
name: VARCHAR2 (100) (nullable)
date_opened: DATE (nullable)
address: VARCHAR2 (200) (nullable)
city: VARCHAR2 (100) (nullable)
zip/postal code: VARCHAR2 (20) (nullable)
phone: VARCHAR2 (20) (nullable)
email: VARCHAR2 (100)
manager_id: NUMBER (10,0) (nullable)
Emergency contact: VARCHAR2 (100) (nullable)

5. Use "(nullable)" to indicate those columns that can have null values.

name (nullable)
date_opened (nullable)
address (nullable)
city (nullable)
zip/postal code (nullable)
phone (nullable)
manager_id (nullable)
Emergency contact (nullable)

6. Write the CREATE TABLE statement for the Global Fast Foods locations table to define the constraints at the column level.

```
CREATE TABLE global_locations (
 Id NUMBER(10,0) PRIMARY KEY,
```

```
 name VARCHAR2(100),
 date_opened DATE,
 address VARCHAR2(200),
 city VARCHAR2(100),
 "zip/postal code" VARCHAR2(20),
 phone VARCHAR2(20),
 email VARCHAR2(100) UNIQUE,
 manager_id NUMBER(10,0),
 "Emergency contact" VARCHAR2(100)
);
```

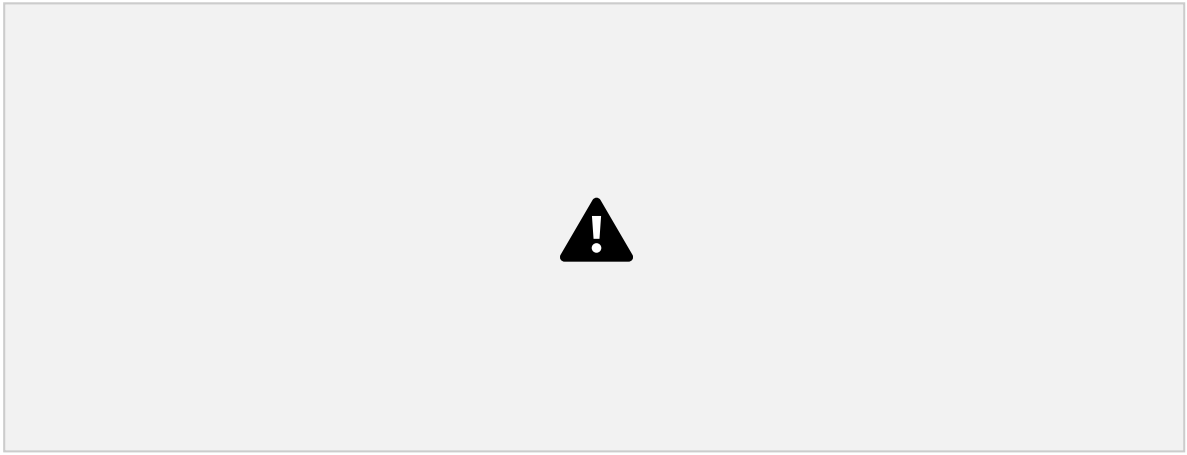
7. Execute the CREATE TABLE statement in Oracle Application Express.

```
CREATE TABLE global_locations (
 Id NUMBER(10,0) PRIMARY KEY,
 name VARCHAR2(100),
 date_opened DATE,
 address VARCHAR2(200),
 city VARCHAR2(100),
 "zip/postal code" VARCHAR2(20),
 phone VARCHAR2(20),
 email VARCHAR2(100) UNIQUE,
 manager_id NUMBER(10,0),
 "Emergency contact" VARCHAR2(100)
);
```

8. Execute a DESCRIBE command to view the Table Summary information.

```
DESCRIBE global_locations;
```

9. Rewrite the CREATE TABLE statement for the Global Fast Foods locations table to define the UNIQUE constraints at the table level. Do not execute this statement.



```
CREATE TABLE global_locations (
 Id NUMBER(10,0) PRIMARY KEY,
 name VARCHAR2(100),
 date_opened DATE,
 address VARCHAR2(200),
 city VARCHAR2(100),
```

```
 "zip/postal code" VARCHAR2(20),
 phone VARCHAR2(20),
 email VARCHAR2(100),
 manager_id NUMBER(10,0),
 "Emergency contact" VARCHAR2(100),
 CONSTRAINT unique_email UNIQUE (email)
);
```

PRIMARY KEY, FOREIGN KEY, and CHECK Constraints

1. What is the purpose of a
- PRIMARY KEY
- FOREIGN KEY
- CHECK CONSTRAINT

PRIMARY KEY: Ensures each record in a table is uniquely identifiable and serves as a unique identifier for each row. It also automatically creates a unique index on the primary key column(s). **FOREIGN KEY**: Maintains referential integrity between two related tables. It establishes a link between a column in one table (child table) and a column in another table (parent table), ensuring that the values in the child table exist in the parent table.

CHECK CONSTRAINT: Validates the values entered into a column to ensure they meet a specific condition or set of conditions. It restricts the range of values that can be entered into a column.

2. Using the column information for the animals table below, name constraints where applicable at the table level, otherwise name them at the column level. Define the primary key (animal_id). The license_tag_number must be unique. The admit_date and vaccination_date columns cannot contain null values.

```
animal_id NUMBER(6)
name VARCHAR2(25)
license_tag_number NUMBER(10)
admit_date DATE
adoption_id NUMBER(5),
vaccination_date DATE
```

- **PRIMARY KEY**: animal_id (column-level)
- **UNIQUE CONSTRAINT**: license_tag_number (column-level)
- **NOT NULL CONSTRAINT**: admit_date, vaccination_date (column-level)

3. Create the animals table. Write the syntax you will use to create the table.

```
CREATE TABLE animals (
 animal_id NUMBER(6) PRIMARY KEY,
 name VARCHAR2(25),
 license_tag_number NUMBER(10) UNIQUE,
 admit_date DATE NOT NULL,
```

adoption_id NUMBER(5),
vaccination_date DATE NOT NULL
);

4. Enter one row into the table. Execute a SELECT * statement to verify your input. Refer to the graphic below for input.

ANIMAL_ID	NAME	LICENSE_TAG_NUMBER	ADMIT_DATE	ADOPTION_ID	VACCINATION_DATE
101	Spot	35540	10-Oct-2004	205	12-Oct-2004

INSERT INTO animals (animal_id, name, license_tag_number, admit_date, adoption_id, vaccination_date)
VALUES (101, 'Spot', 35540, TO_DATE('10-Oct-2004', 'DD-Mon-YYYY'), 205, TO_DATE('12-Oct-2004', 'DD-Mon-YYYY'));

SELECT * FROM animals;

5. Write the syntax to create a foreign key (adoption_id) in the animals table that has a corresponding primary-key reference in the adoptions table. Show both the column-level and table level syntax. Note that because you have not actually created an adoptions table, no adoption_id primary key exists, so the foreign key cannot be added to the animals table.

ALTER TABLE animals
ADD CONSTRAINT fk_adoption_id FOREIGN KEY (adoption_id) REFERENCES adoptions(adoption_id);

ALTER TABLE animals
ADD CONSTRAINT fk_adoption_id FOREIGN KEY (adoption_id)
REFERENCES adoptions(adoption_id);

6. What is the effect of setting the foreign key in the ANIMAL table as:

a. ON DELETE CASCADE
b. ON DELETE SET NULL

ON DELETE CASCADE: If a record in the parent table (adoptions) is deleted, all corresponding records in the child table (animals) will be automatically deleted.
ON DELETE SET NULL: If a record in the parent table (adoptions) is deleted, the corresponding foreign key column values in the child table (animals) will be set to NULL.