PART - A

1. DEPARTMENT (dept_no, dept_name, location)

```
1. Create the Simple DEPARTMENT Table.
2. Display structure of department table.
3. Insert below records into Department Table

4. Display all records of Department table
5. Display all department belonging to location 'NY'
6. Display details of Department 10
7. List all department names starting with 'A'
8. List all departments whose number is between 1 and 100
9. Delete 'TRG' department
10. Change department name 'EDP' to 'IT
```

🔹 DEPARTMENT (dept_no, dept_name, location)

1️⃣ Create the Simple DEPARTMENT Table

```sql
-- Drop the DEPARTMENT table if it already exists
DROP TABLE DEPARTMENT;

-- Create the DEPARTMENT table
CREATE TABLE DEPARTMENT (
    dept_no    NUMBER PRIMARY KEY,   -- Unique identifier for each department
    dept_name  VARCHAR2(30),         -- Name of the department
    location   VARCHAR2(30)          -- Location of the department
);
```

Explanation:

- We **drop** the table first to avoid errors if it already exists.
- `dept_no` is set as the **Primary Key**, meaning it must be **unique** and **not null**.
- `dept_name` and `location` are text fields (`VARCHAR2(30)`).

2️⃣ Display the structure of the department table

```
DESC DEPARTMENT;
```

Expected Output:

Name	Type	Nullable
DEPT_NO	NUMBER	No
DEPT_NAME	VARCHAR2(30)	Yes
LOCATION	VARCHAR2(30)	Yes

3️⃣ Insert the given records into the Department Table

```
INSERT INTO DEPARTMENT VALUES (10, 'Account', 'NY');
INSERT INTO DEPARTMENT VALUES (20, 'HR', 'NY');
INSERT INTO DEPARTMENT VALUES (30, 'Production', 'DL');
INSERT INTO DEPARTMENT VALUES (40, 'Sales', 'NY');
INSERT INTO DEPARTMENT VALUES (50, 'EDP', 'MU');
INSERT INTO DEPARTMENT VALUES (60, 'TRG', 'NY');
INSERT INTO DEPARTMENT VALUES (110, 'RND', 'AH');
```

Explanation:

- Each row represents a department.
- The `dept_no` values are unique, ensuring no **Primary Key violations**.
- Locations are stored using standard city abbreviations (e.g., NY for New York, MU for Mumbai, etc.).

4️⃣ Display all records of the DEPARTMENT table

```
SELECT * FROM DEPARTMENT;
```

Expected Output:

DEPT_NO	DEPT_NAME	LOCATION
10	Account	NY
20	HR	NY
30	Production	DL
40	Sales	NY
50	EDP	MU
60	TRG	NY
110	RND	AH

5️⃣ Display all departments belonging to location 'NY'

```
SELECT * FROM DEPARTMENT WHERE location = 'NY';
```

Expected Output:

DEPT_NO	DEPT_NAME	LOCATION
10	Account	NY
20	HR	NY
40	Sales	NY
60	TRG	NY

6️⃣ Display details of Department 10

```
SELECT * FROM DEPARTMENT WHERE dept_no = 10;
```

Expected Output:

DEPT_NO	DEPT_NAME	LOCATION
10	Account	NY

7️⃣ List all department names starting with 'A'

```
SELECT * FROM DEPARTMENT WHERE dept_name LIKE 'A%';
```

Explanation:

- The `LIKE 'A%'` pattern **retrieves all department names** that **start with 'A'**.
- In this case, **only "Account" starts with 'A'**.

Expected Output:

DEPT_NO	DEPT_NAME	LOCATION
10	Account	NY

8️⃣ List all departments whose number is between 1 and 100

```
SELECT * FROM DEPARTMENT WHERE dept_no BETWEEN 1 AND 100;
```

Expected Output:

DEPT_NO	DEPT_NAME	LOCATION
10	Account	NY
20	HR	NY
30	Production	DL
40	Sales	NY
50	EDP	MU
60	TRG	NY

Explanation:

- The `BETWEEN` clause retrieves departments with numbers **from 1 to 100**.
- Department `110 (RND)` is excluded because it is **greater than 100**.

9️⃣ Delete 'TRG' department

```
DELETE FROM DEPARTMENT WHERE dept_name = 'TRG';
```

Expected Output:

```
1 row deleted.
```

Explanation:

- This removes the **department named 'TRG'** from the table.
- If the `DELETE` statement runs successfully, `TRG` will no longer appear in query results.

🔟 Change department name 'EDP' to 'IT'

```sql
UPDATE DEPARTMENT SET dept_name = 'IT' WHERE dept_name = 'EDP';
```

Expected Output:

```
1 row updated.
```

Explanation:

- The `UPDATE` statement modifies **only the department name**, leaving other fields unchanged.
- The `dept_name` for department `50` is now **'IT'** instead of **'EDP'**.

🛑 Constraint Violation Demonstration

⚠️ Trying to insert a duplicate Primary Key

```sql
INSERT INTO DEPARTMENT VALUES (10, 'Logistics', 'DELHI');
```

🛑 **Oracle Error Output:**

```
ORA-00001: unique constraint (SYS_CXXXXXX) violated
```

Explanation:

- We **cannot insert duplicate** `dept_no = 10` because it is a **Primary Key**, which enforces **uniqueness**.

---

2. EMPLOYEE (emp_id, emp_name, birth_date, gender, dept_no, address, designation, salary,experience, email).

```
DEPARTMENT (dept_no, dept_name, location).
Create the EMP Table with all necessary constraints such as:
In EMP TABLE: Employee id should be primary key, Department no should be Foreign key, employee age (birth_date) should be
greater than 18 years, salary shouldbe greater than zero, email should have (@ and dot) sign in address, designation of
employee can be “manager”, “clerk”, “leader”, “analyst”, “designer”, “coder”,“tester”.
    • Create DEPT table with necessary constraint such as
Department no should be primary key, department name should be unique.
After creation of above tables, modify Employee table by adding the constraints as
‘Male’ or ‘Female’ in gender field and display the structure.
Insert proper data (at least 5 appropriate records) in all the tables.
    1. Describe the structure of table created
    2. List all records of each table in ascending order.
    3. Delete the department whose loction is Ahmedabad.
    4. Display female employee list
    5. Display Departname wise employee Names
    6. Find the names of the employee who has salary less than 5000 and greater than 2000.
    7. Display the names and the designation of all female employee in descending order.
    8. Display the names of all the employees who names starts with ‘A’ ends with ‘A’.
    9. Add 10% raise in salary of all employees whose department is ‘IT’.
```

🔹 EMPLOYEE & DEPARTMENT Table with Constraints – Oracle 11g

🏗️ Step 1: Create the DEPARTMENT Table with Constraints

```
-- Drop the table if it already exists
DROP TABLE DEPARTMENT;

-- Create DEPARTMENT table
CREATE TABLE DEPARTMENT (
    dept_no   NUMBER PRIMARY KEY,     -- Primary key for Department
    dept_name VARCHAR2(50) UNIQUE,    -- Unique constraint ensures department names are distinct
    location  VARCHAR2(50)
);
```

Explanation:

- `dept_no` is the **Primary Key** (ensures uniqueness).
- `dept_name` is **Unique** (no duplicate department names).

🏗️ Step 2: Create the EMPLOYEE Table with Constraints

```
-- Drop the table if it already exists
DROP TABLE EMPLOYEE;

-- Create EMPLOYEE table
CREATE TABLE EMPLOYEE (
    emp_id       NUMBER PRIMARY KEY,            -- Unique Employee ID
    emp_name     VARCHAR2(50) NOT NULL,         -- Employee name cannot be NULL
    birth_date   DATE CHECK (birth_date <= TO_DATE('2007-01-01', 'YYYY-MM-DD')),
    gender       VARCHAR2(10) CHECK (gender IN ('Male', 'Female')),
    dept_no      NUMBER,
    address      VARCHAR2(100),
    designation  VARCHAR2(20) CHECK (designation IN
                 ('manager', 'clerk', 'leader', 'analyst', 'designer', 'coder', 'tester')),
    salary       NUMBER CHECK (salary > 0),     -- Salary must be greater than 0
    experience   NUMBER DEFAULT 0,              -- Default experience is 0
    email        VARCHAR2(100) CHECK (email LIKE '%@%.%'),
    CONSTRAINT fk_dept FOREIGN KEY (dept_no) REFERENCES DEPARTMENT(dept_no) ON DELETE CASCADE
);
```

Explanation:

- `emp_id` **is the Primary Key** (unique and mandatory).
- `dept_no` **is a Foreign Key** referencing the `DEPARTMENT` table (CASCADE on delete).
- `birth_date` **check** ensures the employee is **at least 18 years old**.
- `gender` **constraint** restricts values to `'Male'` or `'Female'`.
- `salary` **check** ensures it is **greater than zero**.
- `email` **check** ensures it contains **'@' and '.'** using `REGEXP_LIKE()`.
- `designation` **constraint** allows only specific job roles.

🛠️ Step 3: Insert Sample Data into DEPARTMENT Table

```
INSERT INTO DEPARTMENT VALUES (101, 'IT', 'NY');
INSERT INTO DEPARTMENT VALUES (102, 'HR', 'DL');
INSERT INTO DEPARTMENT VALUES (103, 'Finance', 'AH');
INSERT INTO DEPARTMENT VALUES (104, 'Sales', 'MU');
INSERT INTO DEPARTMENT VALUES (105, 'Production', 'NY');
```

🛠️ Step 4: Insert Sample Data into EMPLOYEE Table

```
INSERT INTO EMPLOYEE VALUES (1, 'Alice', TO_DATE('2000-05-15', 'YYYY-MM-DD'), 'Female', 101, 'NY, USA', 'coder', 4500, 2,
'alice@example.com');
INSERT INTO EMPLOYEE VALUES (2, 'Bob', TO_DATE('1998-08-21', 'YYYY-MM-DD'), 'Male', 102, 'DL, India', 'manager', 6000, 5,
'bob@example.com');
INSERT INTO EMPLOYEE VALUES (3, 'Charlie', TO_DATE('2001-11-10', 'YYYY-MM-DD'), 'Male', 103, 'AH, India', 'analyst',
3500, 1, 'charlie@example.com');
INSERT INTO EMPLOYEE VALUES (4, 'David', TO_DATE('1997-03-25', 'YYYY-MM-DD'), 'Male', 104, 'MU, India', 'designer', 7500,
7, 'david@example.com');
INSERT INTO EMPLOYEE VALUES (5, 'Emma', TO_DATE('1999-07-30', 'YYYY-MM-DD'), 'Female', 101, 'NY, USA', 'tester', 5200, 3,
'emma@example.com');
```

🔎 Query Solutions

1️⃣ Describe the structure of tables

```
DESC EMPLOYEE;
DESC DEPARTMENT;
```

2️⃣ List all records of each table in ascending order

```
SELECT * FROM EMPLOYEE ORDER BY emp_id ASC;
SELECT * FROM DEPARTMENT ORDER BY dept_no ASC;
```

3️⃣ Delete the department whose location is Ahmedabad

```
DELETE FROM DEPARTMENT WHERE location = 'AH';
```

Explanation:

- If there are **employees linked to `dept_no` in Ahmedabad**, they will also be deleted due to the **ON DELETE CASCADE** constraint.

4️⃣ Display female employee list

```sql
SELECT * FROM EMPLOYEE WHERE gender = 'Female';
```

5️⃣ Display department-wise employee names

```sql
SELECT D.dept_name, E.emp_name
FROM EMPLOYEE E
JOIN DEPARTMENT D ON E.dept_no = D.dept_no
ORDER BY D.dept_name;
```

6️⃣ Find employees with salary between 2000 and 5000

```sql
SELECT emp_name, salary FROM EMPLOYEE WHERE salary BETWEEN 2000 AND 5000;
```

7️⃣ Display female employees' names and designations in descending order

```sql
SELECT emp_name, designation
FROM EMPLOYEE
WHERE gender = 'Female'
ORDER BY emp_name DESC;
```

8️⃣ Find employees whose names start and end with ‘A’

```sql
SELECT emp_name FROM EMPLOYEE WHERE emp_name LIKE 'A%A';
```

9️⃣ Increase salary by 10% for employees in the IT department

```sql
UPDATE EMPLOYEE
SET salary = salary * 1.10
WHERE dept_no = (SELECT dept_no FROM DEPARTMENT WHERE dept_name = 'IT');
```

Explanation:

- The **subquery** fetches the `dept_no` for "IT".
- Employees in IT get **10% salary raise**.

---

3. STUDENT (rollno, name, class, birthdate)

```
COURSE (courseno, coursename, max_marks, pass_marks)
SC (rollno, courseno, marks)
    1. Create the above three tables along with key constraints.
    2. Write an Insert script for insertion of rows with substitution variables and insert appropriate data.
    3. Add a constraint that the marks entered should strictly be between 0 and 100.
    4. While creating SC table, composite key constraint was forgotten. Add the compositekeynow.
    5. Display details of student who takes ‘Database Management System’ course.
    6. Display the names of students who have scored more than 70% in ComputerNetworks and have not failed in any
subject.
    7. Display the average marks obtained by each student.
    8. Select all courses where passing marks are more than 30% of average maximum mark.
```

🔹 STUDENT, COURSE & SC Tables with Constraints – Oracle 11g

🏗️ Step 1: Create Tables with Constraints

1️⃣ Creating the `STUDENT` Table

```
DROP TABLE STUDENT;

CREATE TABLE STUDENT (
    rollno    NUMBER PRIMARY KEY,       -- Unique Roll Number
    name      VARCHAR2(50) NOT NULL,    -- Student Name (Cannot be NULL)
    class     VARCHAR2(20),
    birthdate DATE CHECK (birthdate <= TO_DATE('2007-01-01', 'YYYY-MM-DD')) -- Ensures age is at least 18
);
```

Explanation:

- `rollno` is the **Primary Key** (ensures uniqueness).
- `birthdate` has a **CHECK constraint** to ensure the student is at least **18 years old**.

2️⃣ Creating the `COURSE` Table

```
DROP TABLE COURSE;

CREATE TABLE COURSE (
    courseno   NUMBER PRIMARY KEY,           -- Unique Course Number
    coursename VARCHAR2(100) NOT NULL,       -- Course Name (Mandatory)
    max_marks  NUMBER CHECK (max_marks > 0), -- Maximum marks should be greater than 0
    pass_marks NUMBER CHECK (pass_marks > 0) -- Pass marks must be valid
);
```

Explanation:

- `courseno` is the **Primary Key**.
- **CHECK constraint** ensures:
    - `max_marks` is positive.

- `pass_marks` is **greater than zero** but **less than or equal to** **`max_marks`**.

3️⃣ Creating the `SC` (Student-Course) Table

```
DROP TABLE SC;

CREATE TABLE SC (
    rollno   NUMBER,
    courseno NUMBER,
    marks    NUMBER CHECK (marks BETWEEN 0 AND 100), -- Marks should be between 0 and 100
    CONSTRAINT fk_roll FOREIGN KEY (rollno) REFERENCES STUDENT(rollno) ON DELETE CASCADE,
    CONSTRAINT fk_course FOREIGN KEY (courseno) REFERENCES COURSE(courseno) ON DELETE CASCADE
);
```

Explanation:

- `rollno` is a **Foreign Key** referencing `STUDENT(rollno)`.
- `courseno` is a **Foreign Key** referencing `COURSE(courseno)`.
- **CHECK constraint** ensures `marks` is **between 0 and 100**.

🛠️ Step 2: Insert Data Using Substitution Variables

```
INSERT INTO STUDENT VALUES (&rollno, '&name', '&class', TO_DATE('&birthdate', 'YYYY-MM-DD'));
INSERT INTO COURSE VALUES (&courseno, '&coursename', &max_marks, &pass_marks);
INSERT INTO SC VALUES (&rollno, &courseno, &marks);
```

🔹 **Example Execution:**
When prompted, enter:

- `rollno`: 101

- `name`: 'Alice'
- `class`: 'MCA'
- `birthdate`: '2002-05-10'

🛠️ Step 3: Add Composite Key to `SC` Table

Since we forgot the **composite key**, let's add it now:

```
ALTER TABLE SC ADD CONSTRAINT pk_sc PRIMARY KEY (rollno, courseno);
```

Explanation:

- The **composite key** ensures a **student cannot register for the same course multiple times**.

🔎 Query Solutions

1️⃣ Describe the Structure of Tables

```
DESC STUDENT;
DESC COURSE;
DESC SC;
```

2️⃣ Display Details of Students Taking 'Database Management System'

```
SELECT S.*
FROM STUDENT S
JOIN SC ON S.rollno = SC.rollno
JOIN COURSE C ON SC.courseno = C.courseno
WHERE C.coursename = 'Database Management System';
```

Explanation:

- **Joins all three tables** to get student details for `Database Management System` course.

3️⃣ Display Students with More than 70% in ‘Computer Networks’ and No Fails

```
SELECT S.name
FROM STUDENT S
JOIN SC ON S.rollno = SC.rollno
JOIN COURSE C ON SC.courseno = C.courseno
WHERE C.coursename = 'Computer Networks'
AND SC.marks > (0.7 * C.max_marks)
AND S.rollno NOT IN (
    SELECT rollno FROM SC WHERE marks < (SELECT pass_marks FROM COURSE WHERE COURSE.courseno = SC.courseno)
);
```

Explanation:

- Finds students who scored **more than 70%** in `Computer Networks`.
- Ensures they **haven't failed in any subject**.

4️⃣ Display Average Marks for Each Student

```
SELECT S.rollno, S.name, AVG(SC.marks) AS avg_marks
FROM STUDENT S
JOIN SC ON S.rollno = SC.rollno
GROUP BY S.rollno, S.name;
```

Explanation:

- Groups by `rollno` and `name` to calculate the **average marks** per student.

5️⃣ Select Courses Where Passing Marks > 30% of Average Max Marks

```
SELECT *
FROM COURSE
WHERE pass_marks > (0.3 * (SELECT AVG(max_marks) FROM COURSE));
```

Explanation:

- Calculates **30% of the average `max_marks`** across all courses.
- Selects only courses where `pass_marks` exceed this value.

---

4. Create the database COMPANY and create given tables with all necessary constraints such as primary key, foreign key, unique key, not null and check constraints.

```
EMPLOYEE (emp_id, emp_name, birth_date, gender,
dept_no, address, designation, salary, experience, email)
DEPART (dept_no, dept_name, total_employees, location)
PROJECT (proj_id, type_of_project, status, start_date, emp_id)
Insert proper data (at least 5 appropriate records) in all the tables.

    1. Delete the department whose total number of employees less than 1.
    2. Display the names and the designation of all female employee in descending order.
    3. Display the names of all the employees who names starts with ‘A’ ends with ‘A’.
    4. Find the name of employee and salary for those who had obtain minimum salary.
    5. Add 10% raise in salary of all employees whose department is ‘CIVIL’.
    6. Count total number of employees of ‘MCA’ department.
    7. List all employees who born in the current month.
```

```
8. Print the record of employee and dept table as “Employee works in department ‘CE’.
9. List names of employees who are fresher’s(less than 1 year of experience).
10. List department wise names of employees who has more than 5 years of experience.
```

🔹 COMPANY Database – Table Creation & Queries in Oracle 11g

🏗️ Step 1: Create the Database & Tables

1️⃣ Creating the `DEPART` Table

```
DROP TABLE DEPART;

CREATE TABLE DEPART (
    dept_no         NUMBER PRIMARY KEY,
    dept_name       VARCHAR2(50) UNIQUE NOT NULL,
    total_employees NUMBER CHECK (total_employees >= 0),
    location        VARCHAR2(100)
);
```

Explanation:

- `dept_no` is the **Primary Key**.
- `dept_name` is **Unique** and **Not Null**.
- `total_employees` cannot be **negative**.

2️⃣ Creating the `EMPLOYEE` Table

```
DROP TABLE EMPLOYEE;

CREATE TABLE EMPLOYEE (
    emp_id      NUMBER PRIMARY KEY,
    emp_name    VARCHAR2(50) NOT NULL,
    birth_date  DATE CHECK (birth_date <= TO_DATE('2007-01-01','YYYY-MM-DD')), -- Age must be at least 18
    gender      VARCHAR2(10) CHECK (gender IN ('Male', 'Female')),
    dept_no     NUMBER,
    address     VARCHAR2(100),
    designation VARCHAR2(30) CHECK (designation IN ('Manager', 'Clerk', 'Leader', 'Analyst', 'Designer', 'Coder',
'Tester')),
    salary      NUMBER CHECK (salary > 0),
    experience  NUMBER CHECK (experience >= 0),
    email       VARCHAR2(100) UNIQUE CHECK (email LIKE '%@%.%'),
    CONSTRAINT fk_dept FOREIGN KEY (dept_no) REFERENCES DEPART(dept_no) ON DELETE CASCADE
);
```

Explanation:

- `emp_id` is the **Primary Key**.
- `birth_date` ensures **employee is at least 18 years old**.
- `gender` accepts only **‘Male’ or ‘Female’**.
- `designation` is **restricted to specific job roles**.
- `salary` must be **greater than 0**.
- `experience` cannot be **negative**.
- `email` must contain **‘@’ and ‘.’**.
- `dept_no` is a **Foreign Key** linked to `DEPART`.

3️⃣ Creating the `PROJECT` Table

```
DROP TABLE PROJECT;

CREATE TABLE PROJECT (
    proj_id         NUMBER PRIMARY KEY,
    type_of_project VARCHAR2(50) NOT NULL,
    status          VARCHAR2(30) CHECK (status IN ('Ongoing', 'Completed', 'On-Hold')),
    start_date      DATE NOT NULL,
    emp_id          NUMBER,
    CONSTRAINT fk_emp FOREIGN KEY (emp_id) REFERENCES EMPLOYEE(emp_id) ON DELETE CASCADE
);
```

Explanation:

- `proj_id` is the **Primary Key**.
- `status` must be **‘Ongoing’, ‘Completed’, or ‘On-Hold’**.
- `emp_id` is a **Foreign Key** linked to `EMPLOYEE`.

🛠️ Step 2: Insert Sample Data

```
INSERT INTO DEPART VALUES (1, 'MCA', 5, 'Bangalore');
INSERT INTO DEPART VALUES (2, 'CIVIL', 3, 'Mumbai');
INSERT INTO DEPART VALUES (3, 'IT', 2, 'Delhi');

INSERT INTO EMPLOYEE VALUES (101, 'Alice', TO_DATE('1995-04-10', 'YYYY-MM-DD'), 'Female', 1, 'Bangalore', 'Analyst',
50000, 3, 'alice@mail.com');
INSERT INTO EMPLOYEE VALUES (102, 'Bob', TO_DATE('1990-08-25', 'YYYY-MM-DD'), 'Male', 2, 'Mumbai', 'Manager', 70000, 6,
'bob@mail.com');
INSERT INTO EMPLOYEE VALUES (103, 'Charlie', TO_DATE('2000-01-15', 'YYYY-MM-DD'), 'Male', 1, 'Chennai', 'Tester', 30000,
1, 'charlie@mail.com');
INSERT INTO EMPLOYEE VALUES (104, 'Anna', TO_DATE('2001-07-12', 'YYYY-MM-DD'), 'Female', 3, 'Delhi', 'Designer', 45000,
2, 'anna@mail.com');
INSERT INTO EMPLOYEE VALUES (105, 'David', TO_DATE('1998-11-23', 'YYYY-MM-DD'), 'Male', 2, 'Mumbai', 'Clerk', 25000, 5,
'david@mail.com');

INSERT INTO PROJECT VALUES (201, 'AI Research', 'Ongoing', TO_DATE('2024-01-01', 'YYYY-MM-DD'), 101);
INSERT INTO PROJECT VALUES (202, 'Bridge Construction', 'Completed', TO_DATE('2023-06-15', 'YYYY-MM-DD'), 102);
```

🔎 Step 3: Execute Queries

1️⃣ Delete Departments with Employees Less Than 1

```
DELETE FROM DEPART WHERE total_employees < 1;
```

2️⃣ List Female Employees in Descending Order

```
SELECT emp_name, designation FROM EMPLOYEE WHERE gender = 'Female' ORDER BY emp_name DESC;
```

3️⃣ Employees Whose Name Starts & Ends with ‘A’

```
SELECT emp_name FROM EMPLOYEE WHERE emp_name LIKE 'A%A';
```

4️⃣ Find Employee with Minimum Salary

```
SELECT emp_name, salary FROM EMPLOYEE WHERE salary = (SELECT MIN(salary) FROM EMPLOYEE);
```

5️⃣ Increase Salary by 10% for ‘CIVIL’ Employees

```
UPDATE EMPLOYEE
SET salary = salary * 1.10
WHERE dept_no = (SELECT dept_no FROM DEPART WHERE dept_name = 'CIVIL');
```

6️⃣ Count Employees in ‘MCA’ Department

```
SELECT COUNT(*) FROM EMPLOYEE WHERE dept_no = (SELECT dept_no FROM DEPART WHERE dept_name = 'MCA');
```

7️⃣ List Employees Born in the Current Month

```
SELECT emp_name FROM EMPLOYEE WHERE EXTRACT(MONTH FROM birth_date) = EXTRACT(MONTH FROM SYSDATE);
```

8️⃣ Show Employee with Department Name

```
SELECT emp_name || ' works in department ' || dept_name AS Employee_Department
FROM EMPLOYEE E JOIN DEPART D ON E.dept_no = D.dept_no;
```

9️⃣ List Employees with Less Than 1 Year Experience (Fresher)

```
SELECT emp_name FROM EMPLOYEE WHERE experience < 1;
```

🔟 List Employees with More Than 5 Years Experience (Department-wise)

```
SELECT dept_name, emp_name
FROM EMPLOYEE E
JOIN DEPART D ON E.dept_no = D.dept_no
```

```
WHERE experience > 5
ORDER BY dept_name;
```

---

5. Create the database STUD and create given tables with all necessary constraints such as primary key, foreign key, unique key, not null and check constraints.

```
HOSTEL (HNO, HNAME, HADDR, TOTAL_CAPACITY, WARDEN)
ROOM (HNO, RNO, RTYPE, LOCATION, NO_OF_STUDENTS,
STATUS)CHARGES (HNO, RTYPE, CHARGES)
STUDENT (SID, SNAME, MOBILE-NO, GENDER, FACULTY, DEPT, CLASS,HNO, RNO)
FEES (SID, FDATE, FAMOUNT)
The STATUS field tells us whether the room is occupied or vacant. The charges represent the term fees to be paid half
yearly. A student can pay either the annual fees at one time or the half yearly fees twice a year.
Insert proper data (at least 5 appropriate records) in all the tables.
    1. Display the total number of rooms that are presently vacant.
    2. Display number of students of each faculty and department wise staying in eachhostel.
    3. Display hostels, which have at least one single-seated room.
    4. Display the warden name and hostel address of students of Computer Science department.
    5. Display those hostel details where single seated or double-seated rooms are vacant.
    6. Display details of hostels occupied by medical students.
    7. Display hostels, which are totally occupied to its fullest capacity.
    8. List details about students who are staying in the double- seated rooms of Chanakya Hostel.
    9. Display the total number of students staying in each room type of each hostel.
    10. Display details about students who have paid fees in the month of Nov. 2017.
```

🔹 STUD Database – Oracle 11g Implementation

🏗️ Step 1: Table Creation with Constraints

1️⃣ HOSTEL Table

```sql
DROP TABLE HOSTEL;

CREATE TABLE HOSTEL (
    HNO             NUMBER PRIMARY KEY,
    HNAME           VARCHAR2(50) NOT NULL,
    HADDR           VARCHAR2(100),
    TOTAL_CAPACITY  NUMBER CHECK (TOTAL_CAPACITY > 0),
    WARDEN          VARCHAR2(50) NOT NULL
);
```

2️⃣ ROOM Table

```sql
DROP TABLE ROOM;

CREATE TABLE ROOM (
    HNO             NUMBER,
    RNO             NUMBER,
    RTYPE           VARCHAR2(20) CHECK (RTYPE IN ('Single', 'Double', 'Triple')),
    LOCATION        VARCHAR2(50),
    NO_OF_STUDENTS  NUMBER CHECK (NO_OF_STUDENTS >= 0),
    STATUS          VARCHAR2(10) CHECK (STATUS IN ('Occupied', 'Vacant')),
    PRIMARY KEY (HNO, RNO),
    FOREIGN KEY (HNO) REFERENCES HOSTEL(HNO) ON DELETE CASCADE
);
```

3️⃣ CHARGES Table

```sql
DROP TABLE CHARGES;

CREATE TABLE CHARGES (
    HNO     NUMBER,
    RTYPE   VARCHAR2(20),
    CHARGES NUMBER CHECK (CHARGES > 0),
    PRIMARY KEY (HNO, RTYPE),
    FOREIGN KEY (HNO) REFERENCES HOSTEL(HNO)
);
```

4️⃣ STUDENT Table

```
DROP TABLE STUDENT;

CREATE TABLE STUDENT (
    SID         NUMBER PRIMARY KEY,
    SNAME       VARCHAR2(50) NOT NULL,
    MOBILE_NO   VARCHAR2(10) UNIQUE CHECK (LENGTH(MOBILE_NO) = 10),
    GENDER      VARCHAR2(10) CHECK (GENDER IN ('Male', 'Female')),
    FACULTY     VARCHAR2(50),
    DEPT        VARCHAR2(50),
    CLASS       VARCHAR2(20),
    HNO         NUMBER,
    RNO         NUMBER,
    FOREIGN KEY (HNO, RNO) REFERENCES ROOM(HNO, RNO)
);
```

5️⃣ FEES Table

```
DROP TABLE FEES;

CREATE TABLE FEES (
    SID     NUMBER,
    FDATE   DATE,
    FAMOUNT NUMBER CHECK (FAMOUNT > 0),
    FOREIGN KEY (SID) REFERENCES STUDENT(SID)
);
```

📝 Step 2: Insert Sample Data

HOSTEL

```
INSERT INTO HOSTEL VALUES (1, 'Chanakya', 'Block A', 10, 'Mr. Sharma');
INSERT INTO HOSTEL VALUES (2, 'Ashoka', 'Block B', 8, 'Ms. Latha');
```

ROOM

```
INSERT INTO ROOM VALUES (1, 101, 'Single', 'North Wing', 1, 'Occupied');
INSERT INTO ROOM VALUES (1, 102, 'Double', 'South Wing', 2, 'Occupied');
INSERT INTO ROOM VALUES (1, 103, 'Single', 'East Wing', 0, 'Vacant');
INSERT INTO ROOM VALUES (2, 201, 'Double', 'West Wing', 1, 'Occupied');
INSERT INTO ROOM VALUES (2, 202, 'Triple', 'North Wing', 0, 'Vacant');
```

CHARGES

```
INSERT INTO CHARGES VALUES (1, 'Single', 25000);
INSERT INTO CHARGES VALUES (1, 'Double', 18000);
INSERT INTO CHARGES VALUES (2, 'Double', 15000);
INSERT INTO CHARGES VALUES (2, 'Triple', 12000);
```

STUDENT

```
INSERT INTO STUDENT VALUES (101, 'Arya', '9876543210', 'Female', 'Science', 'Computer Science', 'MCA', 1, 101);
INSERT INTO STUDENT VALUES (102, 'Kiran', '9876500001', 'Male', 'Medical', 'Anatomy', 'MBBS', 1, 102);
INSERT INTO STUDENT VALUES (103, 'Sita', '9876511111', 'Female', 'Science', 'Computer Science', 'MCA', 1, 102);
```

```sql
INSERT INTO STUDENT VALUES (104, 'Rahul', '9876522222', 'Male', 'Medical', 'Pharma', 'BPharm', 2, 201);
INSERT INTO STUDENT VALUES (105, 'Anu', '9876533333', 'Female', 'Arts', 'History', 'BA', 2, 201);
```

FEES

```sql
INSERT INTO FEES VALUES (101, TO_DATE('2023-11-15','YYYY-MM-DD'), 50000);
INSERT INTO FEES VALUES (102, TO_DATE('2023-05-01','YYYY-MM-DD'), 25000);
INSERT INTO FEES VALUES (103, TO_DATE('2023-11-10','YYYY-MM-DD'), 25000);
INSERT INTO FEES VALUES (104, TO_DATE('2023-07-12','YYYY-MM-DD'), 15000);
INSERT INTO FEES VALUES (105, TO_DATE('2023-11-20','YYYY-MM-DD'), 15000);
```

🔍 Step 3: SQL Queries with Explanations

1️⃣ Total Number of Vacant Rooms

```sql
SELECT COUNT(*) AS Vacant_Rooms FROM ROOM WHERE STATUS = 'Vacant';
```

2️⃣ Number of Students Per Faculty, Department, and Hostel

```sql
SELECT HNO, FACULTY, DEPT, COUNT(*) AS Num_Students
FROM STUDENT
GROUP BY HNO, FACULTY, DEPT;
```

3️⃣ Hostels with At Least One Single-Seated Room

```sql
SELECT DISTINCT HNO FROM ROOM WHERE RTYPE = 'Single';
```

4️⃣ Warden and Hostel Address for Computer Science Students

```sql
SELECT DISTINCT H.WARDEN, H.HADDR
FROM STUDENT S
JOIN HOSTEL H ON S.HNO = H.HNO
WHERE S.DEPT = 'Computer Science';
```

5️⃣ Hostel Details with Vacant Single or Double Rooms

```sql
SELECT DISTINCT H.*
FROM HOSTEL H
JOIN ROOM R ON H.HNO = R.HNO
WHERE R.STATUS = 'Vacant' AND R.RTYPE IN ('Single', 'Double');
```

6️⃣ Hostel Details with Medical Students

```sql
SELECT DISTINCT H.*
FROM HOSTEL H
JOIN STUDENT S ON H.HNO = S.HNO
WHERE S.FACULTY = 'Medical';
```

7️⃣ Hostels at Full Capacity

```sql
SELECT H.HNO, H.HNAME, H.TOTAL_CAPACITY, COUNT(S.SID) AS Occupants
FROM HOSTEL H
JOIN STUDENT S ON H.HNO = S.HNO
```

```
GROUP BY H.HNO, H.HNAME, H.TOTAL_CAPACITY
HAVING COUNT(S.SID) = H.TOTAL_CAPACITY;
```

8️⃣ Double-Seated Room Students in Chanakya Hostel

```
SELECT S.*
FROM STUDENT S
JOIN ROOM R ON S.HNO = R.HNO AND S.RNO = R.RNO
JOIN HOSTEL H ON S.HNO = H.HNO
WHERE H.HNAME = 'Chanakya' AND R.RTYPE = 'Double';
```

9️⃣ Total Students by Room Type and Hostel

```
SELECT R.HNO, R.RTYPE, SUM(R.NO_OF_STUDENTS) AS Total_Students
FROM ROOM R
GROUP BY R.HNO, R.RTYPE;
```

🔟 Students Who Paid Fees in Nov 2017

```
SELECT S.*
FROM STUDENT S
JOIN FEES F ON S.SID = F.SID
WHERE TO_CHAR(F.FDATE, 'MM-YYYY') = '11-2017';
```

PART - B

1. Write a PL/SQL to update the rate field by 20% more than the current rate in inventory table which has the following fields: Prono, ProName and Rate. After updating the table a new field (Alter) called for Number of item and place for values for the new field without using PL/SQL block.

🧾 Question Breakdown

We are given a table called `INVENTORY` with the following fields:

- `Prono` (Product Number)
- `ProName` (Product Name)
- `Rate` (Price)

The task is:

1. **Use a PL/SQL block** to:
   - Update the `Rate` by increasing it **20%**.
2. **Then**, add a **new column** called `NumberOfItems` (not using PL/SQL).
3. **Populate** this new column with appropriate values using **a regular SQL statement**, **not** PL/SQL.

🧱 Step 1: Setup the Table and Insert Sample Data (for testing)

```
DROP TABLE INVENTORY;

CREATE TABLE INVENTORY (
    Prono   NUMBER PRIMARY KEY,
    ProName VARCHAR2(50),
    Rate    NUMBER
);
```

```sql
-- Sample Data
INSERT INTO INVENTORY VALUES (1, 'Monitor', 10000);
INSERT INTO INVENTORY VALUES (2, 'Keyboard', 1500);
INSERT INTO INVENTORY VALUES (3, 'Mouse', 800);
INSERT INTO INVENTORY VALUES (4, 'CPU', 25000);

COMMIT;
```

🔁 Step 2: PL/SQL Block to Increase Rate by 20%

```sql
BEGIN
    UPDATE INVENTORY
    SET Rate = Rate + (Rate * 0.20);

    DBMS_OUTPUT.PUT_LINE('Rates updated successfully by 20%!');
END;
/
```

✅ This block increases every product's `Rate` by **20%.**

➕ Step 3: Add a New Column (NOT using PL/SQL)

```sql
ALTER TABLE INVENTORY ADD (NumberOfItems NUMBER);
```

🗃️ Step 4: Populate New Column (Using SQL only)

Let’s assume these are the new stock numbers:

```sql
UPDATE INVENTORY SET NumberOfItems =
    CASE Prono
        WHEN 1 THEN 15
        WHEN 2 THEN 50
        WHEN 3 THEN 75
        WHEN 4 THEN 10
    END;
```

📄 Final Table Output (after all operations)

You can check the final values using:

```sql
SELECT * FROM INVENTORY;
```

✅ Summary

Step	Task	Tool Used
1	Increase rate by 20%	PL/SQL block
2	Add column `NumberOfItems`	SQL (ALTER TABLE)
3	Populate new column	SQL (UPDATE)

---

2. Write a PL/SQL to split the student table into two tables based on result (One table for —Pass‖ and another for —Fail‖). Use cursor for handling records of student table. Assume necessary fields and create a student details table.

🧾 Question:

> **Write a PL/SQL program to split the `STUDENT` table into two tables based on result**
> One table for `'Pass'` and another for `'Fail'`.
> Use **cursor** to handle records from the `STUDENT` table.
> Assume necessary fields and create the `STUDENT_DETAILS` table.

✅ Step-by-Step Solution:

🛠️ Step 1: Create the `STUDENT_DETAILS` Table

We'll assume these fields:

- `roll_no` – number (primary key)
- `name` – varchar2
- `marks` – number
- `result` – varchar2 ('Pass' or 'Fail')

```
DROP TABLE STUDENT_DETAILS;

CREATE TABLE STUDENT_DETAILS (
    roll_no NUMBER PRIMARY KEY,
    name    VARCHAR2(50),
    marks   NUMBER,
    result  VARCHAR2(10)
);
```

➕ Step 2: Insert Sample Records into `STUDENT_DETAILS`

```
INSERT INTO STUDENT_DETAILS VALUES (1, 'Alice', 80, 'Pass');
INSERT INTO STUDENT_DETAILS VALUES (2, 'Bob', 45, 'Fail');
INSERT INTO STUDENT_DETAILS VALUES (3, 'Cathy', 90, 'Pass');
INSERT INTO STUDENT_DETAILS VALUES (4, 'David', 30, 'Fail');
INSERT INTO STUDENT_DETAILS VALUES (5, 'Eva', 70, 'Pass');

COMMIT;
```

🗃️ Step 3: Create `PASS_STUDENTS` and `FAIL_STUDENTS` Tables

```
DROP TABLE PASS_STUDENTS;
DROP TABLE FAIL_STUDENTS;

CREATE TABLE PASS_STUDENTS (
    roll_no NUMBER PRIMARY KEY,
    name    VARCHAR2(50),
    marks   NUMBER
);

CREATE TABLE FAIL_STUDENTS (
    roll_no NUMBER PRIMARY KEY,
    name    VARCHAR2(50),
    marks   NUMBER
);
```

🔁 Step 4: PL/SQL Code Using a CURSOR to Split the Data

```
DECLARE
    CURSOR student_cursor IS
        SELECT roll_no, name, marks, result FROM STUDENT_DETAILS;

    v_rollno STUDENT_DETAILS.roll_no%TYPE;
    v_name   STUDENT_DETAILS.name%TYPE;
    v_marks  STUDENT_DETAILS.marks%TYPE;
    v_result STUDENT_DETAILS.result%TYPE;

BEGIN
    OPEN student_cursor;
    LOOP
        FETCH student_cursor INTO v_rollno, v_name, v_marks, v_result;
        EXIT WHEN student_cursor%NOTFOUND;

        IF v_result = 'Pass' THEN
            INSERT INTO PASS_STUDENTS (roll_no, name, marks)
            VALUES (v_rollno, v_name, v_marks);
        ELSIF v_result = 'Fail' THEN
            INSERT INTO FAIL_STUDENTS (roll_no, name, marks)
            VALUES (v_rollno, v_name, v_marks);
        END IF;

    END LOOP;
    CLOSE student_cursor;

    DBMS_OUTPUT.PUT_LINE('Records split successfully!');
END;
/
```

📊 Step 5: View the Split Tables

```
SELECT * FROM PASS_STUDENTS;
```

```sql
SELECT * FROM FAIL_STUDENTS;
```

✅ Output

PASS_STUDENTS:

roll_no	name	marks
1	Alice	80
3	Cathy	90
5	Eva	70

FAIL_STUDENTS:

roll_no	name	marks
2	Bob	45
4	David	30

---

3. Create a database trigger to implement on master and transaction tables which are based on inventory management system for checking data validity. Assume the necessary fields for both tables. Inventory_Master (product_id, product_name ,stock_quantity ,price_per_unit) Inventory_Transaction (transaction_id, product_id ,transaction_type ,transaction_date)

🧾 Question

> Create a **database trigger** for an Inventory Management System involving a **master** and **transaction** table.
> The trigger must ensure **data validity** during operations.

✅ Objective

We'll assume the following:

- `Inventory_Master` contains **product details** and available **stock**.
- `Inventory_Transaction` logs **purchases (IN)** and **sales (OUT)**.
- The trigger will:
  - **Check if stock is available** when `OUT` transaction is made.
  - **Update stock_quantity** accordingly.

🛠️ Step 1: Create the `Inventory_Master` Table

```sql
DROP TABLE Inventory_Master;

CREATE TABLE Inventory_Master (
    product_id      NUMBER PRIMARY KEY,
    product_name    VARCHAR2(50),
    stock_quantity  NUMBER CHECK (stock_quantity >= 0),
    price_per_unit  NUMBER CHECK (price_per_unit > 0)
);
```

🛠️ Step 2: Create the `Inventory_Transaction` Table

```
DROP TABLE Inventory_Transaction;

CREATE TABLE Inventory_Transaction (
    transaction_id      NUMBER PRIMARY KEY,
    product_id          NUMBER,
    transaction_type    VARCHAR2(10) CHECK (transaction_type IN ('IN', 'OUT')),
    transaction_date    DATE DEFAULT SYSDATE,
    quantity            NUMBER CHECK (quantity > 0),

    CONSTRAINT fk_product
        FOREIGN KEY (product_id) REFERENCES Inventory_Master(product_id)
);
```

📄 Step 3: Insert Sample Data into `Inventory_Master`

```
INSERT INTO Inventory_Master VALUES (1, 'Monitor', 20, 15000);
INSERT INTO Inventory_Master VALUES (2, 'Mouse', 50, 500);
INSERT INTO Inventory_Master VALUES (3, 'Keyboard', 30, 1200);

COMMIT;
```

⚙️ Step 4: Create the Trigger for Stock Validation and Update

```
CREATE OR REPLACE TRIGGER trg_inventory_update
BEFORE INSERT ON Inventory_Transaction
FOR EACH ROW
DECLARE
    v_stock NUMBER;
BEGIN
    -- Get current stock
    SELECT stock_quantity INTO v_stock
    FROM Inventory_Master
    WHERE product_id = :NEW.product_id;

    IF :NEW.transaction_type = 'OUT' THEN
        IF v_stock < :NEW.quantity THEN
            RAISE_APPLICATION_ERROR(-20001, 'Not enough stock available.');
        ELSE
            UPDATE Inventory_Master
            SET stock_quantity = stock_quantity - :NEW.quantity
            WHERE product_id = :NEW.product_id;
        END IF;

    ELSIF :NEW.transaction_type = 'IN' THEN
        UPDATE Inventory_Master
        SET stock_quantity = stock_quantity + :NEW.quantity
        WHERE product_id = :NEW.product_id;
    END IF;
END;
/
```

✅ This trigger ensures:

- You **can't sell more than what's in stock**.
- The master table's stock is automatically **updated**.

🧪 Step 5: Test the Trigger

✔️ Successful IN Transaction

```
INSERT INTO Inventory_Transaction (transaction_id, product_id, transaction_type, quantity)
VALUES (101, 1, 'IN', 5); -- adds 5 to stock
```

✔️ Successful OUT Transaction

```sql
INSERT INTO Inventory_Transaction (transaction_id, product_id, transaction_type, quantity)
VALUES (102, 2, 'OUT', 10); -- subtracts 10 from stock
```

❌ Attempt to Sell More than Stock

```sql
INSERT INTO Inventory_Transaction (transaction_id, product_id, transaction_type, quantity)
VALUES (103, 3, 'OUT', 100); -- will trigger error
```

🔴 **Error**: `ORA-20001: Not enough stock available.`

🔍 Check Stock After Transactions

```sql
SELECT * FROM Inventory_Master;
```
