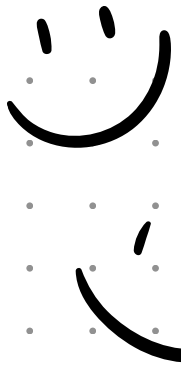

SQL NOTES

BASIC TO ADVANCE CRISP NOTES

PART - 2

@vaibhavsaini

```
----------------------------------------
| Advanced PL/SQL |
----------------------------------------
-- PL/SQL Control Structures:
Control structures like IF-THEN-ELSE and loops provide logical flow control.
Example:
DECLARE
  x NUMBER := 10;
BEGIN
  IF x > 5 THEN
    DBMS_OUTPUT.PUT_LINE('x is greater than 5');
  ELSE
    DBMS_OUTPUT.PUT_LINE('x is not greater than 5');
  END IF;
END;
/

-- Stored Procedures with Parameters:
Create reusable procedures with input parameters.
Example:
CREATE OR REPLACE PROCEDURE UpdateEmployeeSalary(emp_id NUMBER, new_salary NUMBER) AS
BEGIN
  -- Update logic here
END;
/

-- PL/SQL Packages:
Group related procedures, functions, and types for modular code organization.
Example:
CREATE OR REPLACE PACKAGE EmployeeInfo AS
  PROCEDURE GetEmployee(emp_id NUMBER);
  FUNCTION CalculateBonus(salary NUMBER) RETURN NUMBER;
END EmployeeInfo;
/

-- PL/SQL Functions:
Create functions that return values and can be used in SQL expressions.
Example:
CREATE OR REPLACE FUNCTION CalculateTotalCost(price NUMBER, quantity NUMBER) RETURN NUMBER AS
BEGIN
  RETURN price * quantity;
END;
/
```

-- Dynamic SQL:
Build and execute SQL statements dynamically within PL/SQL.
Example:

```
DECLARE
  sql_stmt VARCHAR2(100);
  total_rows NUMBER;
BEGIN
  sql_stmt := 'SELECT COUNT(*) FROM Employees';
  EXECUTE IMMEDIATE sql_stmt INTO total_rows;
  DBMS_OUTPUT.PUT_LINE('Total Rows: ' || total_rows);
END;
/
```

| Normalization and Database Design |

-- Database Design Principles:
Efficient database design is essential for data integrity and performance.
Example:
Consider a company database with tables: Employees, Departments, and Projects.

-- Normalization Forms (1NF, 2NF, 3NF):
Normalization eliminates data redundancy and anomalies.
1. First Normal Form (1NF):
   Each attribute holds only atomic (indivisible) values.
   Example: Employees (EmployeeID, FirstName, LastName, Skills)

2. Second Normal Form (2NF):
   Must satisfy 1NF and have no partial dependencies on a concatenated primary key.
   Example: Employees (EmployeeID, ProjectID, HoursWorked)

3. Third Normal Form (3NF):
   Must satisfy 2NF and have no transitive dependencies.
   Example: Employees (EmployeeID, DepartmentID, ManagerID)

-- Denormalization:
Strategic denormalization enhances query performance.
Example:
Consider a reporting database where data from normalized tables are combined for reporting.

```
----------------------------------------
| Views and Materialized Views |
----------------------------------------
-- Views:
Views create virtual tables by querying data from existing tables.
Example:
CREATE VIEW HighSalaryEmployees AS
SELECT EmployeeName, Salary FROM Employees WHERE Salary > 50000;

-- Materialized Views:
Materialized views store precomputed results for improved performance.
Example:
CREATE MATERIALIZED VIEW MonthlySalesSummary AS
SELECT TRUNC(OrderDate, 'MM') AS Month, SUM(TotalAmount) AS TotalSales
FROM Orders GROUP BY TRUNC(OrderDate, 'MM');

-- Materialized View Refresh:
Refresh materialized views to keep data up to date.
Example:
-- Refresh the materialized view every day at midnight.
BEGIN
  DBMS_SCHEDULER.CREATE_JOB(
    job_name => 'Refresh_MV_Daily',
    job_type => 'PLSQL_BLOCK',
    job_action => 'BEGIN DBMS_MVIEW.REFRESH(''MonthlySalesSummary''); END;',
    start_date => SYSTIMESTAMP,
    repeat_interval => 'FREQ=DAILY; BYHOUR=0; BYMINUTE=0; BYSECOND=0;',
    enabled => TRUE
  );
END;
/
----------------------------------------
| Working with Data Types |
----------------------------------------
-- Common SQL Data Types:
SQL supports various data types to store different kinds of data.
Example:
CREATE TABLE Employees (
 EmployeeID NUMBER,
 FirstName VARCHAR2(50),
 HireDate DATE,
 Salary NUMBER(10, 2)
);
```

-- Handling Date and Time:
Perform operations on date and time values.
Example:

```
SELECT EmployeeName, TO_CHAR(HireDate, 'DD-MON-YYYY') AS HireDate
FROM Employees
WHERE HireDate BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND TO_DATE('2023-03-31', 'YYYY-MM-DD');
```

-- String Manipulation:
Use functions to manipulate and transform string data.
Example:

```
SELECT CustomerName, UPPER(CustomerName) AS UppercaseName, SUBSTR(CustomerName, 1, 3) AS Initials
FROM Customers:
```

| Conclusion |

Working with data types in SQL is fundamental to effectively store, manipulate, and retrieve data. By understanding and utilizing the various data types, you can ensure data integrity and achieve accurate results in your database operations. In this topic, we covered:

- Common SQL data types for different kinds of data.
- Handling date and time values using functions and conversions.
- String manipulation techniques to transform and extract data.

As you continue your SQL journey, mastering data types will empower you to work with diverse data sets and perform intricate calculations with ease. Feel free to explore more advanced topics to enhance your SQL skills further.