SQL: Structured Query Language

- Prepared by: **Prof Momhamad Ubaidullah Bokhari**
- *Department of Computer Science AMU Aligarh*

SQL

- **SQL** stands for **Structured Query Language**
- SQL, Structured Query Language, is the standard language used to communicate with a **relational database**.
- SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc
- It works well on Oracle, Microsoft SQL Server, Sybase, Informix, and others.
- With SQL, you can build databases, enter data into the database, manipulate data, and query the database.
- SQL is a simple, English-like language that is relatively easy to learn.
- It is Case-Insensitive.

Why SQL?

SQL is widely popular because it offers the following advantages :

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

History

- The idea was first suggested by Dr. E.F. Codd's in his research paper.
- The prototype was originally developed by IBM in 1970's.
- In 1979, not long after IBM's prototype, the first SQL product, ORACLE, was released by Relational Software, Incorporated (it was later renamed Oracle Corporation).
- SQL has been deemed the standard language in relational database communication, by American National Standards Institute (ANSI) in 1986. (As character set is an ANSI standard).
- In 1987, the ANSI SQL standard was accepted as the international standard by the International Standards Organization (ISO).
- It was initially called SEQUEL (Structured English QUEry Language). The acronym SEQUEL was later changed to SQL because "SEQUEL" was a trademark of the UK-based aircraft company.

Ways of creating/maintaining database

- Command-line based
- GUI based

SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature –

- DDL - Data Definition Language
- DML - Data Manipulation Language
- DCL - Data Control Language

DDL - Data Definition Language(**Command & Description**)

- **CREATE :**

  Creates a new table, a view of a table, or other object in the database.

- **ALTER :**

  Modifies an existing database object, such as a table.

- **DROP :**

  Deletes an entire table, a view of a table or other objects in the database.

DML - Data Manipulation Language (Command & Description)

- **SELECT :**

  Retrieves certain records from one or more tables.

- **INSERT :**

  Creates a record.

- **UPDATE :**

  Modifies records.

- **DELETE :**

  Deletes records.

DCL - Data Control Language (Command & Description)

- **GRANT**

  Gives a privilege to user.

- **REVOKE**

  Takes back privileges granted from user.

Some important operations

- Creating database
- Inserting values in database
- Extracting/Retrieving data
- Deleting data/table

Attribute Data Types

- **CHAR(size):** character strings of fixed length.
- **VARCHAR(size)/VARCHAR2(size):** character strings of variable length.
- **NUMBER (P, S):** to store fixed or floating point numbers.
  - **P (precision):** max length of data
  - **S (scale):** determines the number of places to the right of the decimal.
- **DATE:** To represent date and time.
- And others

Table Structure

- Department (Table Name)

Dept Name (Character String)	Dept Id (Integer)	Chairman ID (Character String)
Computer Science	007	COMSCIAMU
Chemical Engineering	008	CHESCIAMU
-	-	-
-	-	-
-	-	-

Create Table syntax

- CREATE TABLE Command in SQL-

CREATE TABLE *department*
(Dname ***VARCHAR2(15)*** *,*
Dnumber ***NUMBER,***
Mgr_ssn ***CHAR(9)***
);

Inserting data into Table syntax

- INSERT INTO command-

INSERT INTO *department*

VALUES *('Computer Science', 007, 'COMSCIAMU');*

EXTRACTING DATA FROM TABLE

- SELECT command-

- Extracting whole table
  - **SELECT * FROM** *department*;

- Extracting some columns
  - **SELECT** *Dname* **,** *Mgr_ssn* ***FROM*** *department;*

- Extracting some rows
  - **SELECT * FROM** *department*
    WHERE *Dnumber=007;*

- Extracting some rows and some columns
  - **SELECT** *Dname* **,** *Mgr_ssn* **FROM** department **WHERE** Mgr_ssn='COMSCIAMU'**;**

Deleting records from table

- DELETE FROM command:

- Deleting all records:
  - **DELETE FROM** department;

- Deleting particular rows:
  - **DELETE FROM** department **WHERE** Dnumber=007;

DELETING COMPLETE TABLE

- DROP command-

- **DROP TABLE** department;

Updating existing records

- UPDATE Command:

  - UPDATE department
    SET name='CS', id='010'
    WHERE name='CH';

Modifying the structure of tables

- ALTER TABLE command-
- Adding new columns:
  - ALTER TABLE department ADD (location Varchar2(20));
- Deleting existing columns:
  - ALTER TABLE department DROP COLUMN location;
- Modifying Existing Columns:
  - ALTER TABLE department MODIFY (Dname varchar2(50));

Renaming Table and other commands

- RENAME command:
  - RENAME department TO depart;

- Sorting data in a table:
  - SELECT * FROM depart ORDER BY Dnumber DESC;

- TRUNCATE, DISTINCT, DESCRIBE & others.

Constraints

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

Constraints

- Applying restrictions on data.
- Example:
  - Mobile number should not take characters.
  - Account balance should be more than 1000/-
  - Delivery data can't be less than Order date.

Constraints available in SQL

- NOT NULL constraint.– Ensures that a column cannot have NULL value.
- DEFAULT constraint.- Provides a default value for a column when none is specified.
- Unique Key constraint. Ensures that all values in a column are different.
- Primary Key constraint – Uniquely identifies each row/record in a database table.
- Foreign Key constraint– Uniquely identifies a row/record in any of the given database table.
- Check constraint – The CHECK constraint ensures that all the values in a column satisfies certain conditions.
- Index–Used to create and retrieve data from the database very quickly.

Primary key constraint

- To uniquely identify a row.
  - Example-
    - Enrolment number.
    - PNR status.
    - Order number.
- Associated column values should not be empty.
- A table can have only one primary key.
- It could be composed of:
  - One column
  - More than one columns

Primary key at column level.

- When primary key is composed of only one column.

- CREATE TABLE DEPARTMENT
  (Dname VARCHAR2(25) ,
  Dnumber NUMBER PRIMARY KEY,
  Mgr_ssn CHAR(9)
  );

Composite primary key

Enroll No.	Name	Course	Age
E1	N1	C1	A1
E2	N2	C2	A2
-	-	-	-
-	-	-	-

Composite primary key

Enroll No.	Name	Course	Age
E1	N1	C1	A1
E2	N2	C2	A2
-	-	-	-
-	-	-	-

Enroll No.	Univ. Code	Name	Course	Age
E1	AMU	N1	C1	A1
E2	BHU	N2	C2	A2
E1	BHU	N1	C1	A1
-		-	-	-

Primary key at table level.

- When primary key is composed of more than one columns.
- Also called composite primary key.

- ```
  CREATE TABLE DEPARTMENT
  ( Dname VARCHAR2(25) ,
  Dnumber NUMBER,
  Mgr_ssn CHAR(9),
  PRIMARY KEY (Dnumber, Mgr_ssn)
  );
  ```
  ```

Foreign key

- Consider following table:

Seating Arrangement Table

S. No.	Exam No	Name	Course	Room No
1.	1011	Name1	CS	L-1
2.	1012	Name2	PH	L-1
3.	1013	Name3	CHE	L-3
4.	1014	Name4	CHE	L-4
5.	1015	Name5	CS	L-1

Foreign key

Seating Arrangement Table

S. No.	Exam No	Name	Course	Room No
1.	1011	Name1	CS	L-1
2.	1012	Name2	PH	L-1
3.	1013	Name3	CHE	L-3
4.	1014	Name4	CHE	L-2
5.	1015	Name5	CS	L-1

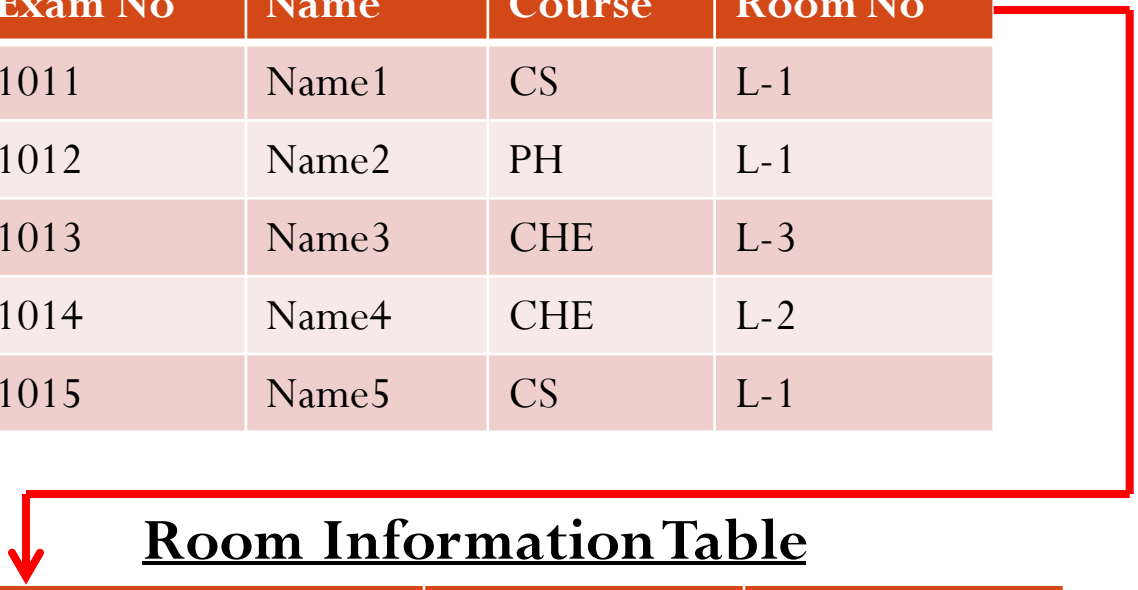
Room Information Table

S. No.	Room No.	Capacity	Invigilator
1.	L-1	30	Invigilator5
2.	L-2	50	Invigilator2
3.	L-3	30	Invigilator1

Foreign key

Seating Arrangement Table

S. No.	Exam No	Name	Course	Room No
1.	1011	Name1	CS	L-1
2.	1012	Name2	PH	L-1
3.	1013	Name3	CHE	L-3
4.	1014	Name4	CHE	L-2
5.	1015	Name5	CS	L-1



Room Information Table

S. No.	Room No.	Capacity	Invigilator
1.	L-1	30	Invigilator5
2.	L-2	50	Invigilator2
3.	L-3	30	Invigilator1

Foreign key

Seating Arrangement Table

S. No.	Exam No	Name	Course	Room No
1.	1011	Name1	CS	L-1
2.	1012	Name2	PH	L-1
3.	1013	Name3	CHE	L-3
4.	1014	Name4	CHE	L-2
5.	1015	Name5	CS	L-1

Room Information Table

S. No.	Room No.	Capacity	Invigilator
1.	**L-1**	30	Invigilator5
2.	**L-2**	50	Invigilator2
3.	**L-3**	30	Invigilator1

Foreign key

Seating Arrangement Table

S. No.	Exam No	Name	Course	Room No
1.	1011	Name1	CS	**L-1**
2.	1012	Name2	PH	**L-1**
3.	1013	Name3	CHE	**L-3**
4.	1014	Name4	CHE	**L-2**
5.	1015	Name5	CS	**L-1**

<-- Foreign

Room Information Table

S. No.	Room No.	Capacity	Invigilator
1.	**L-1**	30	Invigilator5
2.	**L-2**	50	Invigilator2
3.	**L-3**	30	Invigilator1

FOREIGN KEY (ANOTHER EXAMPLE)

Student Information Table

Enroll No.	Name	Course	Age
E1	N1	C1	A1
E2	N2	C2	A2
-	-	-	-
-	-	-	-

FOREIGN KEY (ANOTHER EXAMPLE)

Student Information Table

Enroll No.	Name	Course	Age	Dept Name	Dept Location	Chairman
E1	N1	C1	A1	Dname1	Loc1	C1
E2	N2	C2	A2	Dname1	Loc1	C1
-	-	-	-	-	-	-
-	-	-	-	-	-	-

FOREIGN KEY (ANOTHER EXAMPLE)

Student Information Table

Enroll No.	Name	Course	Age	Dept Name	Dept Location	Chairman
E1	N1	C1	A1	Dname1	Loc1	C1
E2	N2	C2	A2	Dname1	Loc1	C1
-	-	-	-	-	-	-
-	-	-	-	-	-	-

Redundancy → Data Inconsistency , wastage of storage

FOREIGN KEY (ANOTHER EXAMPLE)

Student Information Table

Enroll No.	Name	Course	Age
E1	N1	C1	A1
E2	N2	C2	A2
-	-	-	-
-	-	-	-

Department Information Table

Dept Name	Dept Location	Chairman
Dname1	Loc1	C1
Dname2	Loc2	C2
-	-	-

FOREIGN KEY (ANOTHER EXAMPLE)

Student Information Table

Enroll No.	Name	Course	Age	DNo
E1	N1	C1	A1	D1
E2	N2	C2	A2	D1
-	-	-	-	
-	-	-	-	

Department Information Table

DNum	Dept Name	Dept Location	Chairman
D1	Dname1	Loc1	C1
D2	Dname2	Loc2	C2
-	-	-	-

FOREIGN KEY (ANOTHER EXAMPLE)

Student Information Table

Enroll No.	Name	Course	Age	DNo
E1	N1	C1	A1	**D1**
E2	N2	C2	A2	**D1**
-	-	-	-	
-	-	-	-	

Department Information Table

DNum	Dept Name	Dept Location	Chairman
D1	Dname1	Loc1	C1
D2	Dname2	Loc2	C2
-	-	-	-

FOREIGN KEY (ANOTHER EXAMPLE)

Student Information Table

Enroll No.	Name	Course	Age	DNo
E1	N1	C1	A1	**D1**
E2	N2	C2	A2	**D1**
-	-	-	-	
-	-	-	-	

← Foreign

Department Information Table

Primary →

DNum	Dept Name	Dept Location	Chairman
D1	Dname1	Loc1	C1
D2	Dname2	Loc2	C2
-	-	-	-

Foreign key

- represents relationship between tables.
- references to some other table.
- It could be composed of:
  - One column
  - More than one columns

Foreign key

- At column level-

- CREATE TABLE student
  (Senroll VARCHAR2(10) PRIMARY KEY,
  Dno NUMBER REFERENCES department (Dnumber),
  Sname VARCHAR2(25)
  );

Foreign key

- At table level-

- CREATE TABLE student
  (Senroll VARCHAR2(10) PRIMARY KEY,
  Dno NUMBER,
  Sname VARCHAR2(25),
  FOREIGN KEY (Dno) REFERENCES department (Dnumber)
  );

Unique column constraint

- Column should not have duplicate values.
- It could be null.

- Command:
- CREATE TABLE student
  (Senroll VARCHAR2(10) PRIMARY KEY,
  Dno NUMBER REFERENCES department (Dnumber),
  Sname VARCHAR2(25),
  Voter_id VARCHAR2(25) **UNIQUE**
  );

NOT NULL constraint

- CREATE TABLE student
  (Senroll VARCHAR2(10) PRIMARY KEY,
  Dno NUMBER REFERENCES department (Dnumber),
  Sname VARCHAR2(25) **NOT NULL**,
  Voter_id VARCHAR2(25) UNIQUE
  );

CHECK constraint

- To apply business rule validations:
  - Account balance should be more than 1000/-
  - Delivery data can't be less than Order date.

- CREATE TABLE student
  (Senroll VARCHAR2(10) **CHECK (Senroll LIKE 'S%')**,
  Dno NUMBER,
  Sname VARCHAR2(25),
  Voter_id VARCHAR2(25)
  );

DEFAULT values

- When the user does not provide any value, some 'default' value should take its value.

- CREATE TABLE student
  (Senroll VARCHAR2(10),
  Dno NUMBER **DEFAULT** 008,
  Sname VARCHAR2(25),
  Voter_id VARCHAR2(25)
  );

CURSORS

- The oracle engine uses a work area for its internal processing in order to execute an SQL statement. This work area is called a CURSOR.
- The data that is stored in the cursor is called Active data-set.
- SELECT ClientNo, Cname FROM Client_Master;

ClientNo	Cname
C0001	Ivan
C0002	Bill

Active data-set

Types of cursor

- Implicit-
  - Cursor name- SQL
  - Managed by oracle engine, so no need to define by user.
  - Its attributes are used to access information about last insert, update, delete etc.

- Explicit-
  - Defined by user.
  - Also has same attributes.
  - Useful when individual records have to be processed.

Cursor attributes

Attribute Name	Description
%ISOPEN	Returns TRUE if cursor is open, FALSE otherwise
%FOUND	Returns TRUE if record was fetched successfully, FALSE otherwise.
%NOTFOUND	Returns TRUE if record was not fetched successfully, FALSE otherwise.
%ROWCOUNT	Returns number of records processed.

Implicit cursor- SQL%FOUND and SQL%NOTFOUND


```
/* implicit cursor SQL%FOUND and SQL%NOTFOUND */
declare
    macc varchar2(10);
begin
    macc := &macc;
    update acct_mstr set curbal = 20000 where accountno = macc;
    if SQL%FOUND then
        dbms_output.put_line('Balance updated');
    end if;
    if SQL%NOTFOUND then
        dbms_output.put_line('No such record');
    end if;
end;
/
```

Implicit cursor- SQL%ROWCOUNT


```
/* implicit cursor ROWCOUNT */
declare
    macc varchar2(10);
begin
    macc := &macc;
    update acct_mstr set curbal = 20000 where accountno = macc;
    if SQL%ROWCOUNT >0 then
        dbms_output.put_line(to_char(SQL%ROWCOUNT)||' row(s) is(are) updated');
    else
        dbms_output.put_line('No record was updated');
    end if;
end;
/
```

Explicit cursor

- In PL/SQL, accepting input from the user is not a common situation.
- The data to be processed is taken from existing tables.
- Some mechanism is required to process individual records.
- Explicit cursor serves the purpose.

Explicit cursor

Col_1	Col_2	Col_3
-	-	-
-	-	-
-	-	-
-	-	-
-	-	-

Explicit cursor

Col_1	Col_2	Col_3
-	-	-
-	-	-
-	-	-
-	-	-
-	-	-

Explicit cursor

Col_1	Col_2	Col_3
-	-	-
-	-	-
-	-	-
-	-	-
-	-	-

Explicit cursor

Col_1	Col_2	Col_3
-	-	-
-	-	-
-	-	-
-	-	-
-	-	-

Explicit cursor

Col_1	Col_2	Col_3
-	-	-
-	-	-
-	-	-
-	-	-
-	-	-

Explicit cursor

- Three commands are used to control explicit cursor:
  - **Open**: starts a cursor.
  - **Fetch**: extracts records one by one.
  - **Close**: closes a cursor.

- Attributes:
  - %ISOPEN
  - **%**FOUND
  - %NOTFOUND
  - %ROWCOUNT

Example

- If no transaction took place in an account from last one year then set its status as inactive and record its account no in INACTV_ACCT_MSTR.


```
declare
    cursor cus1 is select accountno, ltd from acct_mstr where (sysdate - ltd)>365;
    macc acct_mstr.accountno%type;
    tdate acct_mstr.ltd%type;
begin









    CLOSE cus1;
end;
/
```

Example

- If no transaction took place in an account from last one year then set its status as inactive and record its account no in INACTV_ACCT_MSTR.


```
declare
    cursor cus1 is select accountno, ltd from acct_mstr where (sysdate - ltd)>365;
    macc acct_mstr.accountno%type;
    tdate acct_mstr.ltd%type;
begin
    OPEN cus1;
    if cus1%ISOPEN then
        
        
        
        
        
        
        
        
    else
        dbms_output.put_line('Problem with cursor');
    end if;
    CLOSE cus1;
end;
/
```

Example

- If no transaction took place in an account from last one year then set its status as inactive and record its account no in INACTV_ACCT_MSTR.


```
declare
    cursor cus1 is select accountno, ltd from acct_mstr where (sysdate - ltd)>365;
    macc acct_mstr.accountno%type;
    tdate acct_mstr.ltd%type;
begin
    OPEN cus1;
    if cus1%ISOPEN then
        loop
            fetch cus1 into macc, tdate;
            exit when cus1%NOTFOUND;



        end loop;
    else
        dbms_output.put_line('Problem with cursor');
    end if;
    CLOSE cus1;
end;
/
```

Example

- If no transaction took place in an account from last one year then set its status as inactive and record its account no in INACTV_ACCT_MSTR.


```
declare
    cursor cus1 is select accountno, ltd from acct_mstr where (sysdate - ltd)>365;
    macc acct_mstr.accountno%type;
    tdate acct_mstr.ltd%type;
begin
    OPEN cus1;
    if cus1%ISOPEN then
        loop
            fetch cus1 into macc, tdate;
            exit when cus1%NOTFOUND;

            if cus1%FOUND then
                update acct_mstr set status = 'Inactive' where accountno = macc;
                insert into inactv_acct_mstr values (macc);
            end if;
        end loop;
    else
        dbms_output.put_line('Problem with cursor');
    end if;
    CLOSE cus1;
end;
/
```

Example

- List first three highest balance accounts.


```
declare
    cursor cus2 is select accountno, curbal from acct_mstr order by curbal desc;
    macc acct_mstr.accountno%type;
    bal acct_mstr.curbal%type;
begin




end;
/
```

Example

- List first three highest balance accounts.


```
declare
    cursor cus2 is select accountno, curbal from acct_mstr order by curbal desc;
    macc acct_mstr.accountno%type;
    bal acct_mstr.curbal%type;
begin
    open cus2;




    close cus2;
end;
/
```

Example

- List first three highest balance accounts.


```
declare
    cursor cus2 is select accountno, curbal from acct_mstr order by curbal desc;
    macc acct_mstr.accountno%type;
    bal acct_mstr.curbal%type;
begin
    open cus2;
    loop
        fetch cus2 into macc, bal;


    end loop;
    close cus2;
end;
/
```

Example

- List first three highest balance accounts.


```
declare
    cursor cus2 is select accountno, curbal from acct_mstr order by curbal desc;
    macc acct_mstr.accountno%type;
    bal acct_mstr.curbal%type;
begin
    open cus2;
    loop
        fetch cus2 into macc, bal;
        exit when ((cus2%ROWCOUNT>3) or (cus2%NOTFOUND));
            dbms_output.put_line(macc || ' '||bal);
    end loop;
    close cus2;
end;
/
```

Procedure and function

- A procedure or function is a logically grouped set of PL/SQL statements that perform a specific task.
- They are compiled and stored separately.
- Similar to PL/SQL block.
- They also have declarative, executable and exception-handling section.

Function- Structure

- Similar to PL/SQL block except:
  - Has **Input Parameters.**
  - Has **Return** keyword.
  - No **Declare** keyword.

```
create or replace function multBy2(num in number) return number is
begin
    return (num*2);
end;
/

select multBy2(curbal) from acct_mstr;
```

Example- function

```
/* Write a function to calculate factorial of a number*/
create or replace function fact(num in number) return number is
f number:=1;
begin
    for c in 1..num
    loop
        f := f*c;
    end loop;
    return f;
end;
/

declare
    n number;
begin
    n := &n;
    dbms_output.put_line('Factorial of ' || to_char(n) || ' is '||to_char(fact(n)));
end;
/
```

Procedure- Structure

- Similar to PL/SQL block except:
  - Has **Input and Output Parameters:**
    - In
    - Out
    - In Out
  - No **Declare** keyword.
  - No **Return** keyword.

Example- Procedure

```
create or replace procedure multBy2p(num in number, m out number) is
begin
    m:= num*2;
end;
/

declare
    i number;
    o number;
begin
    i := &i;
    multBy2p(i,o);
    dbms_output.put_line('Multiple of ' || to_char(i) || ' is '||to_char(o));
end;
/
```

Example- Procedure

```
/* Write a procedure to calculate factorial of a number*/
create or replace procedure factp(num in number, f out number) is
begin
    f:=1;
    for c in 1..num
    loop
        f := f*c;
    end loop;
end;
/

declare
    n number;
    fa number:=0;
begin
    n := &n;
    factp(n,fa);
    dbms_output.put_line('Factorial of ' || to_char(n) || ' is '||to_char(fa));
end;
/
```

Example- Procedure

```
create or replace procedure factp1(num in out number) is
f number:=1;
begin
    for c in 1..num
    loop
        f := f*c;
    end loop;
    num:= f;
end;
/

declare
    n number;
begin
    n := &n;
    factp1(n);
    dbms_output.put_line('Factorial is ' || to_char(n));
end;
/
```

Procedure vs Function

- A function returns a value, a procedure does not.

- Procedure has out parameters, function does not.

Procedure vs Function

- A function returns a value, a procedure does not.
- Procedure has out parameters, function does not.
- Procedures are useful when multiple values are to be ***returned***.
- Functions are useful since they can be used in SQL queries.

Example

```
create table stuMarks
(
Rollno number,
markMath number,
markHindi number,
markEng number
);

insert all
into stuMarks values (1, 80,76,45)
into stuMarks values (2, 42,65,78)
into stuMarks values (3, 54,85,98)
select * from dual;

create table avgMarks
(
Rollno number,
avgno number
);
```

Example

```
create function avg3(num1 in number, num2 in number, num3 in number) return number is
a number;
begin
    a := (num1+num2+num3)/3;
    return a;
end;
/

create procedure insertInAvg(roll in number, aMarks in number) is
begin
    insert into avgMarks values(roll, aMarks);
end;
/

declare
    cursor avgCus is select Rollno, markMath, markHindi, markEng from stuMarks;
    aMarks number;
    roll stuMarks.Rollno%type;
    mark1 stuMarks.markMath%type;
    mark2 stuMarks.markHindi%type;
    mark3 stuMarks.markEng%type;
begin
    open avgCus;
    loop
        fetch avgCus into roll, mark1, mark2, mark3;
        exit when avgCus%NOTFOUND;
        aMarks := avg3(mark1, mark2, mark3);
        insertInAvg(roll,aMarks);
    end loop;
    close avgCus;
end;
/
```

Deleting Function & Procedure

- DROP PROCEDURE prodedure_name;
- DROP FUNCTION function_name;

Trigger

- It is a procedure that is implicitly executed on some designated event.

- Advantages:
  - To prevent invalid transactions.
    - Withdrawal amount can not be greater than balance.
  - To keep audit details.
    - Before deleting a record, its details should be stored in some other table.

Types of triggers

- Based on affecting entity.
- Row trigger:
  - A row trigger is fired each time a row in a table is affected.
    - Before deleting each record, its details should be stored in some other table.
- Statement Trigger:
  - A statement trigger is fired, independent of the number of rows affected.
    - When deletion takes place in a table, store the deletion time in a separate table.

Types of triggers

- Based on trigger timing.
- Before trigger:
  - Executed before the triggering statement.
    - Before withdrawing an amount check whether user has sufficient balance.
- After trigger:
  - Executed after the triggering statement.
    - After deletion takes place in a table, store the deletion time in a separate table.
- Before and After apply to both row and statement triggers.

:NEW & :OLD

- A Trigger has access to the NEW and OLD values of various operations.
  - Update operation:
    - :NEW
    - :OLD
  - Insert operation:
    - :NEW
  - Delete operation:
    - :OLD

Example

```
/* A trigger to calculate Average marks*/
create table stuAvg
(
Rollno number primary key,
markMath number,
markHindi number,
markEng number,
avgMarks number
);
```

Example

```
/* A trigger to calculate Average marks*/
create table stuAvg
(
Rollno number primary key,
markMath number,
markHindi number,
markEng number,
avgMarks number
);
```

```
insert into stuAvg (Rollno,markMath,markHindi,markEng) values (1, 10, 20,30);
```

Example

```
/* A trigger to calculate Average marks*/
create table stuAvg
(
Rollno number primary key,
markMath number,
markHindi number,
markEng number,
avgMarks number
);

create or replace trigger tgrAvg before insert on stuAvg
for each row
begin

end;
/

insert into stuAvg (Rollno,markMath,markHindi,markEng) values (1, 10, 20,30);
```

Example

```
/* A trigger to calculate Average marks*/
create table stuAvg
(
Rollno number primary key,
markMath number,
markHindi number,
markEng number,
avgMarks number
);

create or replace trigger tgrAvg before insert on stuAvg
for each row
begin
    :NEW.avgMarks := (:NEW.markMath + :NEW.markHindi +:NEW.markEng)/3;
end;
/

insert into stuAvg (Rollno,markMath,markHindi,markEng) values (1, 10, 20,30);
```

Example

```
/* When an account is updated then insert the affected rows and corresponding operation in a new table*/
 create table rec_acc
(
acc varchar2(7),
op varchar2(10),
opdate date
 );
```

Example

```
/* When an account is updated then insert the affected rows and corresponding operation in a new table*/
 create table rec_acc
(
acc varchar2(7),
op varchar2(10),
opdate date
 );
```

```
update acct_mstr set status = 'Active' where status = 'Inactive';
delete from acct_mstr where curbal = 2000;
```

Example

```
/* When an account is updated then insert the affected rows and corresponding operation in a new table*/
 create table rec_acc
(
acc varchar2(7),
op varchar2(10),
opdate date
 );

create or replace trigger tgrRec after update or delete on acct_mstr for each row
declare
op varchar2(10);
begin









end;
/

update acct_mstr set status = 'Active' where status = 'Inactive';
delete from acct_mstr where curbal = 2000;
```

Example

```
/* When an account is updated then insert the affected rows and corresponding operation in a new table*/
 create table rec_acc
(
acc varchar2(7),
op varchar2(10),
opdate date
 );

create or replace trigger tgrRec after update or delete on acct_mstr for each row
declare
op varchar2(10);
begin
    if updating then
        op := 'UPDATE';
    end if;
    if deleting then
        op := 'DELETE';
    end if;

end;
/

update acct_mstr set status = 'Active' where status = 'Inactive';
delete from acct_mstr where curbal = 2000;
```

Example

```
/* When an account is updated then insert the affected rows and corresponding operation in a new table*/
 create table rec_acc
(
acc varchar2(7),
op varchar2(10),
opdate date
 );

create or replace trigger tgrRec after update or delete on acct_mstr for each row
declare
op varchar2(10);
begin
    if updating then
        op := 'UPDATE';
    end if;
    if deleting then
        op := 'DELETE';
    end if;
    insert into rec_acc values (:old.accountno, op,sysdate);
end;
/

update acct_mstr set status = 'Active' where status = 'Inactive';
delete from acct_mstr where curbal = 2000;
```

Example

```
/* implementing foreign key using trigger */
create table dept
(
 Dno number,
 Dname varchar2(5)
);
```

```
create table emp
(
 Eno varchar2(5),
 Dnum number
 );
```

```
insert all
into dept values (10, 'CS')
into dept values (20, 'MATH')
select * from dual;
```

Example

```
/* implementing foreign key using trigger */
create table dept
(
 Dno number,
 Dname varchar2(5)
);
```

```
create table emp
(
 Eno varchar2(5),
 Dnum number
 );
```

```
insert all
into dept values (10, 'CS')
into dept values (20, 'MATH')
select * from dual;
```

```
insert into emp values ('E1', 40);
```

Example

```
/* implementing foreign key using trigger */
create table dept
(
 Dno number,
 Dname varchar2(5)
);

create table emp
(
 Eno varchar2(5),
 Dnum number
 );

insert all
into dept values (10, 'CS')
into dept values (20, 'MATH')
select * from dual;

create or replace trigger tgrChkIns before insert on emp for each row
declare
chk varchar2(4);
begin



end;
/

insert into emp values ('E1', 40);
```

Example

```
/* implementing foreign key using trigger */
create table dept
(
 Dno number,
 Dname varchar2(5)
);

create table emp
(
 Eno varchar2(5),
 Dnum number
 );

insert all
into dept values (10, 'CS')
into dept values (20, 'MATH')
select * from dual;

create or replace trigger tgrChkIns before insert on emp for each row
declare
chk varchar2(4);
begin
    select 'Y' into chk from dept where Dno = :new.Dnum;



end;
/

insert into emp values ('E1', 40);
```

Example

```
/* implementing foreign key using trigger */
create table dept
(
 Dno number,
 Dname varchar2(5)
);

create table emp
(
 Eno varchar2(5),
 Dnum number
 );

insert all
into dept values (10, 'CS')
into dept values (20, 'MATH')
select * from dual;

create or replace trigger tgrChkIns before insert on emp for each row
declare
chk varchar2(4);
begin
    select 'Y' into chk from dept where Dno = :new.Dnum;
    exception
        when no_data_found then

end;
/

insert into emp values ('E1', 40);
```

Example

```
/* implementing foreign key using trigger */
create table dept
(
 Dno number,
 Dname varchar2(5)
);

create table emp
(
 Eno varchar2(5),
 Dnum number
 );

insert all
into dept values (10, 'CS')
into dept values (20, 'MATH')
select * from dual;

create or replace trigger tgrChkIns before insert on emp for each row
declare
chk varchar2(4);
begin
    select 'Y' into chk from dept where Dno = :new.Dnum;
    exception
        when no_data_found then
        raise_application_error(-20002,'Such department does not exist in DEPT table');
end;
/

insert into emp values ('E1', 40);
```

Additional concepts

- RAISE_APPLICATION_ERROR:
  - Stops execution of current operation.
  - Issues user defined error message.
  - Error number should be in the range -20000 to -20999.

- Enable and Disable trigger:
  - ALTER TRIGGER tgrRec DISABLE;
  - ALTER TRIGGER tgrRec ENABLE;

- Trigger can be applied on particular column:
  - CREATE OR REPLACE TRIGGER tgrSalDes AFTER UPDATE OF salary, designation ON employee

    . . . . . .

    . . . . . .

Additional concepts

- Difference between Procedure and Trigger:
  - Triggers do not accept parameters, Procedures do.
  - Triggers are called implicitly, Procedures are called explicitly.

- Triggers vs Integrity constraints:
  - Both are used to implement constraints.
  - Similarity:
    - Implicitly called.
  - Difference:
    - Integrity constraints are usually defined during table creation and applicable on whole data. While triggers does not apply on already loaded data.
    - A trigger enforces a constraint which can not be enforced by integrity constraint.

Deleting a trigger

- DROP TRIGGER trigger_name;

Resources

- Software:
  - Oracle 11g express edition
  - Freely available

- Book:
  - SQL, PL/SQL by **Ivan Bayross**