### Mysql - <https://dev.mysql.com/downloads/installer/>

Mongodb – <https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-6.0.3.zip>

Mongodb shell - <https://downloads.mongodb.com/compass/mongodb-mongosh_1.6.0_amd64.deb>

Both workbench and shell

### SQL ass 1:

### SQL Queries:

### • Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as

### (Table, View, Index, Sequence, Synonym, different constraints etc.)

| Sr. No. | Key | DDL | DML |
| --- | --- | --- | --- |
| 1 | Stands for | DDL stands for Data Definition Language. | DML stands for Data Manipulation Language. |
| 2 | Usage | DDL statements are used to create database, schema, constraints, users, tables etc. | DML statement is used to insert, update or delete the records. |
| 3 | Classification | DDL has no further classification. | DML is further classified into procedural DML and non-procedural DML. |
| 4 | Commands | CREATE, DROP, RENAME and ALTER. | INSERT, UPDATE and DELETE. |

|  |  |  |
| --- | --- | --- |
| Effect | The DDL command affects the entire database or table. | The DML commands will affect the single or multiple records based on the specified condition. |

### (NOTE: cannot create synonym for table but for database in Mysql)

### 

### (Note: MUL means foreign key that references to em\_id of employee)

### CREATE TABLE employee(

### em\_id INT,

### em\_name CHAR(20),

### em\_join\_date DATE,

### PRIMARY key(em\_id)

### );

### DESC employee;

### CREATE TABLE emp\_finance(

### pay\_id INT auto\_increment,

### em\_id INT,

### em\_salary INT,

### em\_contract DATE,

### FOREIGN key(em\_id) REFERENCES employee(em\_id),

### PRIMARY key(pay\_id)

### );

### DESC emp\_finance;

### INSERT INTO employee VALUES(1, 'sham', '2020-09-22');

### INSERT INTO employee VALUES(2, 'ram', '2020-09-12');

### INSERT INTO employee VALUES(3, 'ram', '2020-09-15');

### SELECT \* FROM employee;

### INSERT INTO emp\_finance VALUES(1,1,2000,'2021-01-21');

### INSERT INTO emp\_finance VALUES(null,3,4000,'2021-04-21');

### SELECT \* FROM emp\_finance;

### CREATE VIEW EmpDetails AS

### SELECT e.em\_id, e.em\_name, f.em\_salary FROM employee AS e

### INNER JOIN emp\_finance AS f

### WHERE e.em\_id = f.em\_id;

### SELECT \* FROM EmpDetails;

### ALTER TABLE emp\_finance ADD bonus int(10);

### CALL sys.create\_synonym\_db('ass1', 'assignment1');

### show databases;

### use assignment1;

### show tables;

### 

### • Write at least 10 SQL queries on the suitable database application using SQL DML statements.

### (Insert, Select, Update, Delete with operators, Functions, and set operator)

### Insert, select, update, delete

### Using

### Operators -> in, not in, limit, is null, like, set, between

### Functions ->

### Strings : upper, lower, length, concat, substring

### Numeric: abs, mod, sqrt, power

### Date: curdate, curtime, year, month, day

### Aggregate: avg, sum, min, max, count

### Create database ass1b;

### Use ass1b;

### CREATE TABLE emp(emp\_id int primary key auto\_increment,

### emp\_fname char(20) not null, emp\_lname char(20) not null,

### emp\_salary int, join\_date date);

### Desc emp;

### INSERT INTO emp values(1, 'Ram', 'Kapoor', 40000, '2019-02-23');

### INSERT INTO emp values(null, 'Ramesh', 'Kapoor', 20000, '2019-02-24');

### INSERT INTO emp values(null, 'Ramu', 'Kapoor', 4000, '2019-02-25');

### Select \* FROM emp;

### UPDATE emp set emp\_salary=10000 where emp\_id=3;

### DELETE FROM emp where emp\_id=3;

### SELECT AVG(emp\_salary) FROM emp;

### SELECT concat(emp\_fname, emp\_lname) FROM emp where emp\_salary in (SELECT max(emp\_salary) FROM emp);

### SELECT upper(concat(emp\_fname, emp\_lname)) FROM emp where emp\_salary in (SELECT max(emp\_salary) FROM emp);

### SELECT \* FROM emp WHERE YEAR(join\_date)='2015';

### SELECT \* FROM emp WHERE emp\_name like 'Ram%';

### SELECT \* FROM emp WHERE emp\_id in (1,2,3);

### SELECT \* FROM emp WHERE emp\_id not in (1,2,3);

### SELECT \* FROM emp WHERE YEAR(join\_date) between '2015' and '2018';

### 

### SQL Assignment 2:

### SQL Queries – all types of Join, Sub-Query and View:

### Write at least10 SQL queries for suitable database application using SQL DML statements. Note: Instructor will design the queries which demonstrate the use of concepts like

### all types of Join ,Sub-Query and View

### An **SQL Join statement** is used to combine data or rows from two or more tables based on a common field between them.

### A **subquery** is a query that is nested inside a SELECT, INSERT, UPDATE, or DELETE statement, or inside another subquery.

### Joins and subqueries are both used to combine data from different tables into a single result.

### What is a Subquery?

### A *subquery* is a nested query (inner query) that’s used to filter the results of the outer query. Subqueries can be used as an alternative to joins. A subquery is typically nested inside the WHERE clause.

### Syntax:

### SELECT <columns>

### FROM <table>

### WHERE <column> <operator>

### (SELECT <columns>

### FROM <table>

### )

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

* (INNER) JOIN: Returns records that have matching values in both tables
* LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
* RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
* FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

      

(NOTE: Mysql doesn't support full joins)

### Create database ass2;

### Use ass2;

### Create table student(roll\_no int primary key, name char(20), address char(40), phone int(10), age int(2));

### Desc student;

### INSERT INTO student values(1,'Harsh', 'delhi', '11', '18');

### INSERT INTO student values(2,'Harshal', 'mumbai', '12', '16');

### INSERT INTO student values(3,'prathamesh ', 'pune', '13', '20');

### INSERT INTO student values(4,'Shreyash ', 'chennai', '14', '21');

### INSERT INTO student values(3,'prathamesh ', 'pune', '13', '20');

### INSERT INTO student values(4,'Shreyash ', 'chennai', '14', '21');

### INSERT INTO student values(5,'pra', 'pune', '13', '20');

### INSERT INTO student values(6,'Shr', 'chennai', '14', '21');

### CREATE TABLE studentcourse (cid int(1), roll\_no int, foreign key(roll\_no) references student(roll\_no));

### Desc studentcourse;

### INSERT INTO studentcourse values(1,2);

### INSERT INTO studentcourse values(2,1);

### INSERT INTO studentcourse values(3,3);

### INSERT INTO studentcourse values(4,4);

### SELECT \* FROM student

### Inner join studentcourse

### ON student.roll\_no = studentcourse.roll\_no;

### SELECT \* FROM student

### left join studentcourse

### ON student.roll\_no = studentcourse.roll\_no;

### SELECT \* FROM student

### Right join studentcourse

### ON student.roll\_no = studentcourse.roll\_no;

### SELECT \* FROM student

### join studentcourse;

### SELECT \* FROM student;

### CREATE VIEW newView AS

### SELECT student.name, student.roll\_no, studentcourse.cid FROM student

### LEFT JOIN studentcourse

### ON student.roll\_no = studentcourse.roll\_no;

### SELECT \* FROM newView;

### SELECT \* FROM student WHERE roll\_no in

### (SELECT roll\_no FROM studentcourse);

### SELECT \* FROM student

### WHERE roll\_no in

### (SELECT roll\_no FROM studentcourse

### WHERE student.age < 20);

### 

### MongoDB Queries:

### Design and Develop MongoDB Queries using CRUD operations. (Use CRUD

### operations, SAVE method,

### CRUD OPERATIONS: Create, Read, Update and Delete

|  |  |
| --- | --- |
| MongoDB | RDBMS |
| It is a non-relational and document-oriented database. | It is a relational database. |
| It is suitable for hierarchical data storage. | It is not suitable for hierarchical data storage. |
| It has a dynamic schema. | It has a predefined schema. |
| It centers around the CAP theorem (Consistency, Availability, and Partition tolerance). | It centers around ACID properties (Atomicity, Consistency, Isolation, and Durability). |
| In terms of performance, it is much faster than RDBMS. | In terms of performance, it is slower than MongoDB. |

Document Oriented: In MongoDB, all the data stored in the documents instead of tables like in RDBMS. In these documents, the data is stored in fields(key-value pair) instead of rows and columns which make the data much more flexible in comparison to RDBMS. And each document contains its unique object id.

Advantage - It provides great flexibility to the fields in the documents.

Disadvantage - It uses high memory for data storage.

|  |  |
| --- | --- |
| Method | Description |
| db.collection.insert() | It is used to insert a single document in the collection. |
| db.createCollection() | It is used to create an empty collection. |

It's not db.library.delete it's remove

### db.createCollection("employee");

### db.employee.insert({"id":1,"name":"C++"});

### db.employee.insert({"id":2,"name":"Java"});

### db.employee.insert({"id":3,"name":"Python"});

### db.employee.find();

### db.employee.update({"name":"Python"},{$set:{"name":"Python.3.7"}})

### db.employee.find();

### db.employee.remove({"id":1});

### db.employee.find();

### db.employee.find({"name":"Java"});

### db.employee.insert({"id":4,"name":"Java","desc":"dummy" });

### db.employee.find({$and:[{"name":"Java"},{"desc":"dummy"}]});

### db.employee.find({$or:[{"name":"Java"},{"desc":"dummy"}]});

### db.employee.find({"bid":{$in:[1,3]}})

### db.employee.find({"bid":{$nin:[1,3]}})

### db.employee.save({"\_id":ObjectId("123456789abcdef123456789"),"bid":6,"name":"save query","cost":1000})

### 

### Unnamed PL/SQLcode block: Use of Control structure and Exception handling is

### mandatory.

### Suggested Problem statement:

### Consider Tables:

### 1. Borrower (Roll\_no, Name, Date\_of\_Issue, Name\_of\_Book, Status)

### 2. Fine (Roll\_no, Date, Amt)

###  Accept Roll\_no and Name\_of\_Book from user.

###  Check the number of days (from Date\_of\_Issue).

###  If days are between 15 to 30 then fine amount will be Rs 5per day.

###  If no. of days>30, per day fine will be Rs 50 per day and for days less than 30,

### Rs. 5 per day.

###  After submitting the book, status will change from I to R.

###  If condition of fine is true, then details will be stored into fine table.

###  Also handles the exception by named exception handler or user define

### exception handler

**SQL -> Declarative (Tells what to do)**

**PL-SQL (Procedural SQL) -> (What to do + How to do)**

**PL -SQL Block/Code**

Declaration

a int

b int

c int

Executable code

begin

a:=10;

b:=10;

c:= a + b;

end

Exception Handling

Error

|  |  |
| --- | --- |
| **S.No** | **Parameter Mode & Description** |
| 1 | **IN**  An IN parameter lets you pass a value to the subprogram. **It is a read-only parameter**. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value; however, in that case, it is omitted from the subprogram call. **It is the default mode of parameter passing. Parameters are passed by reference**. |
| 2 | **OUT**  An OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. **The actual parameter must be variable and it is passed by value**. |

* **Functions** − These subprograms return a single value; mainly used to compute and return a value.
* **Procedures** − These subprograms do not return a value directly; mainly used to perform an action.

Parts of a PL/SQL Subprogram

Each PL/SQL subprogram has a name, and may also have a parameter list. Like anonymous PL/SQL blocks, the named blocks will also have the following three parts −

|  |  |
| --- | --- |
| **S.No** | **Parts & Description** |
| 1 | **Declarative Part**  It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution. |
| 2 | **Executable Part**  This is a mandatory part and contains statements that perform the designated action. |
| 3 | **Exception-handling**  This is again an optional part. It contains the code that handles run-time errors. |

**DECLARE**

   << declaration **section** >>

**BEGIN**

   <<Block **of** executable code>>

**EXCEPTION**

       << exception handling >>

**WHEN** excp1 **THEN**

      << excp1 handling block >>

**WHEN** excp2  **THEN**

      << excp2 handling block >>

   ........

**WHEN** others **THEN**

    << excp2 handling block>>

**END**;

### create database ass4;

### use ass4;

### Create table borrower(rollin int(11) not null primary key, name char(20) not null, dateofIssue date, bname char(20), status char(1));

### Create table fine(rollno int(11), fdate date, amt int(11), foreign key(rollno) references borrower(rollin));

### desc borrower;

### desc fine;

### INSERT INTO borrower values(1,'a', '2022-11-01', 'java', 'I');

### INSERT INTO borrower values(2,'b', '2022-10-01', 'networking', 'I');

### INSERT INTO borrower values(3,'c', '2022-09-01', 'DBMS', 'I');

### INSERT INTO borrower values(4,'d', '2022-08-01', 'CN', 'I');

### SELECT \* FROM fine;

### (we use delimiter to push our code in block and not one by one)

### delimiter $

### CREATE procedure fine\_calculation (IN rno int(3), bname char(20))

### begin

### Declare i\_date date;

### Declare diff int;

### Declare fine\_amt int;

### Declare exit handler for sqlexception select 'Table not found';

### SELECT dateofIssue into i\_date from borrower where rollin=rno and bname=bname;

### Select datediff(curdate(), i\_date) into diff;

### If(diff>15 and diff<=30) then

### Set fine\_amt = diff\*5;

### Insert into fine values(rno, curdate(), fine\_amt);

### Elseif(diff>30) then

### Set fine\_amt=15\*5 + (diff-30)\*50;

### Insert into fine values(rno, curdate(),fine\_amt);

### End if;

### Update borrower set status='R' where rollin=rno and bname=bname;

### End

### $

### Delimiter ;

### Call fine\_calculation(3, 'DBMS');

### SELECT \* FROM fine;

### SELECT \* FROM borrower;

### (NOTE: here rno in rollin = rno and bname in bname = name are formal parameters of the procedure)

### (At last the delimiter is changed back to ; as it was $ first)

### 

### Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)

### Write a PL/SQL block of code using parameterized Cursor that will merge the data

### available in the newly created table N\_Roll\_Call with the data available in the table

### O\_Roll\_Call. If the data in the first table already exists in the second table then that

### data should be skipped.

|  |  |
| --- | --- |
| Implicit Cursors | Explicit Cursors |
| Implicit cursors are automatically created when select statements are executed. | Explicit cursors needs to be defined explicitly by the user by providing a name. |
| They are capable of fetching a single row at a time. | Explicit cursors can fetch multiple rows. |
| Closes automatically after execution. | Need to close after execution. |
| They are more vulnerable to errors such as Data errors, etc. | They are less vulnerable to errors(Data errors etc.) |
| Provides less programmatic control to the users | User/Programmer has the entire control. |
| Implicit cursors are less efficient. | Comparative to Implicit cursors, explicit cursors are more efficient. |
| Implicit Cursors are defined as:   BEGIN  SELECT attr\_name from table\_name  where CONDITION;  END | Explicit cursors are defined as:   DECLARE  CURSOR cur\_name IS  SELECT attr\_name from table\_name  where CONDITION;  BEGIN  ... |
| Implicit cursors requires anonymous buffer memory for storage purpose. | Explicit cursors use user-defined memory space for storage purpose |
| Cursor attributes use prefix “SQL”.  Structure for implicit cursors: SQL%attr\_name  Few implicit cursors attributes are: SQL%FOUND, SQL%NOTFOUND, SQL%ROWCOUNT | Structure for explicit cursors: cur\_name%attr\_name  Few explicit cursors are: cur\_name%FOUND, cur\_name%NOTFOUND, cur\_name%ROWCOUNT |

### 

### Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)

### Write a PL/SQL block of code using parameterized Cursor that will merge the data

### available in the newly created table N\_Roll\_Call with the data available in the table

### O\_Roll\_Call. If the data in the first table already exists in the second table then that

### data should be skipped.

### Create table o\_rollcall(rno int(11) not null primary key, name varchar(20), addr varchar(20));

### Create table n\_rollcall(rno int(11), name varchar(20), addr varchar(20));

### Desc o\_rollcall;

### Desc n\_rollcall;

### INSERT INTO o\_rollcall VALUES( 1,'sunny','pune');

### INSERT INTO o\_rollcall VALUES( 2,'bunny','chennai ');

### INSERT INTO o\_rollcall VALUES( 3,'lunny','mumbai ');

### INSERT INTO o\_rollcall VALUES( 4,'dunny','delhi ');

### INSERT INTO o\_rollcall VALUES( 5,'raunny','kolkata');

SELECT \* FROM o\_rollcall;

### INSERT INTO n\_rollcall VALUES( 1,'sunny','pune');

### INSERT INTO n\_rollcall VALUES( 2,'bunny','chennai ');

### INSERT INTO n\_rollcall VALUES( 3,'lunny','mumbai ');

SELECT \* FROM n\_rollcall;

declare exit\_cond boolean;

declare continue handler for not found set exit\_cond = TRUE;

### Delimiter $

### Create procedure n11(IN rno1 int)

### Begin

### Declare rno2 int;

### Declare exit\_cond Boolean;

### Declare c1 cursor for select rno from o\_rollcall where rno>rno1;

### Declare continue handler for not found set exit\_cond=TRUE;

### Open c1;

### L1: loop

### Fetch c1 into rno2;

### If not exists(select \* from n\_rollcall where rno=rno2)

### then

### Insert into n\_rollcall select \* from o\_rollcall where rno=rno2;

### End if;

### If exit\_cond then

### Close c1;

### Leave l1;

### End if;

### End loop l1;

### End;

### $

### Delimiter ;

### Call n11(3);

### Select \* from n\_rollcall;

### 