

Society for Computer Technology & Research’s

**PUNE INSTITUTE OF COMPUTER TECHNOLOGY**

Database Management System

Laboratory Manual

#### 

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **Name of Assignment** | **Page No.** | **Date** | **Remark** |
|  | **Group A - Database Programming Languages – SQL, PL/SQL** | | |  |
| **1** | ER Modeling and Normalization: |  |  |  |
| **2** | **SQL Queries**   1. Design and Develop SQLDDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym, different constraints etc. 2. Write at least 10 SQL queries on the suitable database application using SQL DML statements |  |  |  |
| **3** | **SQL Queries all types of Join, Sub-Query and View:**  Write at least10 SQL queries for suitable database application using SQL DML statements |  |  |  |
| **4** | **Unnamed PL/SQL code block:** Use of Control structure and Exception handling is mandatory. Write a PL/SQL block of code for the following requirements:-  Schema:   1. **Borrower**(Roll,Name,DateofIssue, NameofBook, Status) 2. **Fine** (Roll, Date, Amt)    Accept Roll & N ame 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 &  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.  **Frame the problem statement for writing PL/SQL block**  **inline with above statement.** |  |  |  |
| **5** | **Named PL/SQL Block: PL/SQL Stored Procedure and Stored Function.**  Write a Stored Procedure namely proc\_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks899and 825 category is Higher Second Class.  Write a PL/SQLblock to use procedure created with above requirement.  Stud\_Marks(name, total\_marks) Result(Roll,Name, Class) |  |  |  |
| **6** | **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\_EmpId with the data available in the table O\_EmpId.  If the data in the first table already exist in the second table then that data should be skipped. |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | **Database Trigger (All Types: Row level and Statement**  **level triggers, Before and After Triggers).** Write a database trigger on **Library** table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in **Library\_Audit** table.  **Frame problem statement for writing Database Triggers of all types, inline with above statement. The problem statement should clearly state the requirements.** |  |  |  |
| **8** | **Database Connectivity:**  Write a program to implement MySQL/Oracle database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.) |  |  |  |
|  | **Group B: NoSQL Databases** | | |  |
| **9** | Design and Develop MongoDB Queries using **CRUD operations**. (Use CRUD operations, SAVE method, logical operators) |  |  |  |
| **10** | **MongoDB Aggregation and Indexing:**  Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB |  |  |  |
| **11** | **MongoDB Map-reduces operations:**  Implement Map reduces operation with suitable example using MongoDB. |  |  |  |
| **12** | **Database Connectivity:**  Write a program to implement Mongo DB database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.) |  |  |  |
|  | **Group C Mini Project : Database Project Life Cycle** | | |  |
| **13** | Using the **database concepts covered in Group A and Group B**, develop an application with following details:   1. Follow the same problem statement decided in Assignment -1 of Group A. 2. Follow the Software Development Life cycle and other concepts learnt in **Software Engineering Course** throughout the implementation. 3. Develop application considering:  * Front End: Java/Perl/PHP/Python/Ruby/.net/any other language   Backend : MongoDB/ MySQL/Oracle. |  |  |  |

|  |  |
| --- | --- |
| **Assignment No.** | **1** |
| **Title** | ER Modeling and Normalization |
| **PROBLEM STATEMENT/DEFINITION** | Decide a case study related to real time application in group of 2-3 students and formulate a problem statement for application to be developed. Propose a Conceptual Design using ER features using tools like ERD plus, ER Win etc. (Identifying entities, relationships between entities, attributes, keys, cardinalities, generalization, specialization etc.) Convert the ER diagram into relational tables and normalize Relational data model. |
| **Objectives** | a**)** Data Modelingb)converting ERD to table c) Explore ant ERD Tools |
| **Software packages and hardware apparatus used** | MySQL/Oracle  PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse |
| **References** | **Database Management System Laboratory** |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | * Date * Title * Problem Definition * Learning Objective * Learning Outcome * Theory-Related concept,Architecture,Syntax etc * Class Diagram/ER diagram * Test cases * Program Listing * Output * Conclusion |

### Assignment No. 1

**Title :** ER Modeling and Normalization:

**Objectives :** a**)** Data Modelingb)converting ERD to table c) ERD Tools

##### Theory : Reduction of ER diagram to tables

**Introduction** The database can be represented using the notations, and these notations can be reduced to a collection of tables. In the database, every entity set or relationship set can be represented in tabular form.

**The ER diagram is given below:**

##### DBMS Reduction of ER diagram to Table

##### There are some points for converting the ER diagram to the table:

* **Entity type becomes a table.**

In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.

* **All single-valued attribute becomes a column for the table.**

In the STUDENT entity, STUDENT\_NAME and STUDENT\_ID form the column of STUDENT table. Similarly, COURSE\_NAME and COURSE\_ID form the column of COURSE table and so on.

**A key attribute of the entity type represented by the primary key.**

##### In the given ER diagram, COURSE\_ID, STUDENT\_ID, SUBJECT\_ID, and LECTURE\_ID are the key attribute of the entity.

* **The multivalued attribute is represented by a separate table.**

##### In the student table, a hobby is a multivalued attribute. So it is not possible to represent multiple values in a single column of STUDENT table. Hence we create a table STUD\_HOBBY with column name STUDENT\_ID and HOBBY. Using both the column, we create a composite key.

##### Composite attribute represented by components

##### In the given ER diagram, student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET, and STATE. In the STUDENT table, these attributes can merge as an individual column.

* **Derived attributes are not considered in the table.**

##### In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.

##### Using these rules, you can convert the ER diagram to tables and columns and assign the mapping between the tables. Table structure for the given ER diagram is as below:

##### C:\Users\PICT\Desktop\dbms-reduction-of-er-diagram-into-table2.png

##### RDBMS Terminology:

Before we proceed to explain database system, let's revise few definitions related to database.

**Database:** A database is a collection of tables, with related data.

**Table:** A table is a matrix with data. A table in a database looks like a simple spreadsheet. **Column**: One column (data element) contains data of one and the same kind, for example the Column postcode.

**Row**: A row (tuple, entry or record) is a group of related data, for example the data of one subscription.

**Redundancy**: Storing data twice, redundantly to make the system faster.

**Primary Key**: A primary key is unique. A key value cannot occur twice in one table. With a key, you can find at most one row.

**Foreign Key**: A foreign key is the linking pin between two tables.

**Compound Key**: A compound key(composite key) is a key that consists of multiple columns, Because one column is not sufficiently unique.

**Index**: An index in a database resembles an index at the back of a book.

**Referential Integrity**: Referential Integrity makes sure that a foreign key value always points to An existing row.

is a fast, easy-to-use RDBMS being used for many small and big businesses. is developed, marketed, and supported by My SQLAB, which is a Swedish company. is becoming so popular because of many good reasons:

* is released under an open-source license. So you have nothing to pay to use it.
* is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
* uses a standard form of the well-known SQL data language.
* works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
* works very quickly and works well even with large datasets.
* is very friendly to PHP, the most appreciated language for web development.
* supports large databases, upto 50 million rows or more in a table.

The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million tera bytes(TB).

* is customizable. The open-source GPL license allows programmers to modify the

software to fit their own specific environments.

##### The SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

##### Syntax

CREATE TABLE *table\_name* (

*column1 datatype*, *column2 datatype*, *column3 datatype*, );

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

##### SQL CREATE TABLE Example

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

##### Example

CREATE TABLE Persons ( PersonID int,

LastName varchar(255), FirstName varchar(255), Address varchar(255), City varchar(255) );

##### Create Table Using Another Table

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement.

The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

##### Syntax

CREATE TABLE *new\_table\_name* AS SELECT *column1, column2,...*

FROM *existing\_table\_name*

WHERE ;

##### SQL General Data Types

Each column in a database table is required to have a name and a data type.

SQL developers have to decide what types of data will be stored inside each and every table column when creating a SQL table. The data type is a label and a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

The following table lists the general data types in SQL:

|  |  |
| --- | --- |
| **Data type** | **Description** |
| CHARACTER(n) | Character string. Fixed-length n |
| VARCHAR(n) or CHARACTER  VARYING(n) | Character string. Variable length. Maximum length n |
| BINARY(n) | Binary string. Fixed-length n |
| BOOLEAN | Stores TRUE or FALSE values |
| VARBINARY(n) or  BINARY VARYING(n) | Binary string. Variable length. Maximum length n |
| INTEGER(p) | Integer numerical (no decimal). Precision p |
| SMALLINT | Integer numerical (no decimal). Precision 5 |
| INTEGER | Integer numerical (no decimal). Precision 10 |
| BIGINT | Integer numerical (no decimal). Precision 19 |

|  |  |
| --- | --- |
| DECIMAL(p,s) | Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the  decimal |
| NUMERIC(p,s) | Exact numerical, precision p, scale s. (Same as DECIMAL) |
| FLOAT(p) | Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a  single number specifying the minimum precision |
| REAL | Approximate numerical, mantissa precision 7 |
| FLOAT | Approximate numerical, mantissa precision 16 |
| DOUBLE PRECISION | Approximate numerical, mantissa precision 16 |
| DATE | Stores year, month, and day values |
| TIME | Stores hour, minute, and second values |
| TIMESTAMP | Stores year, month, day, hour, minute, and second values |
| INTERVAL | Composed of a number of integer fields, representing a period of time, depending on the type of interval |
| ARRAY | A set-length and ordered collection of elements |
| MULTISET | A variable-length and unordered collection of elements |
| XML | Stores XML data |

##### The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

##### INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways.

The first way specifies both the column names and the values to be inserted: INSERT INTO *table\_name* (*column1*, *column2*, *column3*, ...)

VALUES (*value1*, *value2*, *value3*, ...);

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

INSERT INTO *table\_name* VALUES (*value1*, *value2*, *value3*, ...); The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

##### SELECT Syntax

SELECT *column1*, *column2, ...* FROM *table\_name*;

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

SELECT \* FROM *table\_name*;

##### The SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

* The AND operator displays a record if all the conditions separated by AND is TRUE.
* The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

##### AND Syntax

SELECT *column1*, *column2, ...* FROM *table\_name*

WHERE *condition1* AND *condition2* AND *condition3 ...*;

##### OR Syntax

SELECT *column1*, *column2, ...* FROM *table\_name*

WHERE *condition1* OR *condition2* OR *condition3 ...*;

##### NOT Syntax

SELECT *column1*, *column2, ...* FROM *table\_name*

WHERE NOT *condition*;

**AND Example :** The following SQL statement selects all fields from "Customers" where country is "Germany" AND city is "Berlin":

##### Example

SELECT \* FROM Customers

WHERE Country='Germany' AND City='Berlin';

**OR Example :** The following SQL statement selects all fields from "Customers" where city is "Berlin" OR "München":

##### Example

SELECT \* FROM Customers

WHERE City='Berlin' OR City='München';

**NOT Example :** The following SQL statement selects all fields from "Customers" where country is NOT "Germany":

##### Example

SELECT \* FROM Customers WHERE NOT Country='Germany';

##### The SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

##### ORDER BY Syntax

SELECT *column1*, *column2, ...*

FROM *table\_name*

ORDER BY *column1, column2, ...* ASC|DESC;

##### ORDER BY Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:

##### Example

SELECT \* FROM Customers ORDER BY Country;

##### The SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values. Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values. The SELECT DISTINCT statement is used to return only distinct (different) values.

##### SELECT DISTINCT Syntax

SELECT DISTINCT *column1*, *column2, ...* FROM *table\_name*;

##### SELECT DISTINCT Examples

The following SQL statement selects only the DISTINCT values from the "Country" column in the "Customers" table:

##### Example

SELECT DISTINCT Country FROM Customers;

##### The SQL WHERE Clause

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

##### WHERE Syntax

SELECT *column1*, *column2, ..* FROM *table\_name* WHERE *condition*;

##### WHERE Clause Example

The following SQL statement selects all the customers from the country "Mexico", in the "Customers" table:

##### Example

SELECT \* FROM Customers WHERE Country='Mexico';

##### Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

##### Example

SELECT \* FROM Customers WHERE CustomerID=1;

##### Operators in The WHERE Clause

The following operators can be used in the WHERE clause:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| = | Equal |
| <> | Not equal. **Note:** In some versions of SQL this operator may be written as != |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |
| <= | Less than or equal |
| BETWEEN | Between an inclusive range |
| LIKE | Search for a pattern |
| IN | To specify multiple possible values for a column |

##### The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

##### DELETE Syntax

DELETE FROM *table\_name* WHERE *condition*;

**Note:** Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) that should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

##### SQL DELETE Example

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

##### Example

DELETE FROM Customers

WHERE CustomerName='Alfreds Futterkiste';

##### Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM *table\_name*;

##### The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column. The MAX() function returns the largest value of the selected column. **MIN() Syntax**

SELECT MIN(*column\_name*) FROM *table\_name*

WHERE *condition*;

##### MAX() Syntax

SELECT MAX(*column\_name*) FROM *table\_name* WHERE *condition*;

##### MIN() Example

The following SQL statement finds the price of the cheapest product:

##### Example

SELECT MIN(Price) AS SmallestPrice FROM Products;

##### MAX() Example

The following SQL statement finds the price of the most expensive product:

##### Example

SELECT MAX(Price) AS LargestPrice FROM Products;

##### The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criteria. The AVG() function returns the average value of a numeric column.

The SUM() function returns the total sum of a numeric column.

##### COUNT() Syntax

SELECT COUNT(*column\_name*) FROM *table\_name*

WHERE *condition*;

##### AVG() Syntax

SELECT AVG(*column\_name*) FROM *table\_name*

WHERE *condition*;

##### SUM() Syntax

SELECT SUM(*column\_name*) FROM *table\_name*

WHERE *condition*;

##### COUNT() Example

The following SQL statement finds the number of products:

##### Example

SELECT COUNT(ProductID) FROM Products;

##### AVG() Example

The following SQL statement finds the average price of all products:

##### Example

SELECT AVG(Price) FROM Products;

##### SUM() Example

The following SQL statement finds the sum of the "Quantity" fields in the "OrderDetails" table:

##### Example

SELECT SUM(Quantity) FROM OrderDetails;

##### The SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. There are two wildcards used in conjunction with the LIKE operator:

* % - The percent sign represents zero, one, or multiple characters
* \_ - The underscore represents a single character

The percent ( % ) sign and the underscore ( \_ )can also be used in combinations!

##### LIKE Syntax

SELECT *column1, column2, ...* FROM *table\_name*

WHERE *columnN* LIKE *pattern*;

**Tip:** You can also combine any number of conditions using AND or OR operators. Here are some examples showing different LIKE operators with '%' and '\_' wildcards:

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that starts with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that ends with "a" |
| WHERE CustomerName LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE CustomerName LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE CustomerName LIKE  'a\_%\_%' | Finds any values that starts with "a" and are at least 3  characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that starts with "a" and ends with "o" |

##### SQL LIKE Examples

The following SQL statement selects all customers with a CustomerName starting with "a":

##### Example

SELECT \* FROM Customers WHERE CustomerName LIKE 'a%';

The following SQL statement selects all customers with a CustomerName ending with "a":

##### Example

SELECT \* FROM Customers WHERE CustomerName LIKE '%a';

The following SQL statement selects all customers with a CustomerName that have "or" in any position:

##### Example

SELECT \* FROM Customers

WHERE CustomerName LIKE '%or%';

The following SQL statement selects all customers with a CustomerName that have "r" in the second position:

##### Example

SELECT \* FROM Customers WHERE CustomerName LIKE '\_r%';

The following SQL statement selects all customers with a Customer Name that starts with "a" and are at least 3 characters in length:

##### Example

SELECT \* FROM Customers

WHERE CustomerName LIKE 'a\_%\_%';

The following SQL statement selects all customers with a Customer Name that starts with "a" and ends with "o":

##### Example

SELECT \* FROM Customers WHERE ContactName LIKE 'a%o';

The following SQL statement selects all customers with a CustomerName that NOT starts with "a":

##### Example

SELECT \* FROM Customers

WHERE CustomerName NOT LIKE 'a%';

**Conclusion:** Thus we have studied how to use open source database .

### FAQ ?

1. Compare Vs. SQL Server.
2. What are the features of ?
3. What do DDL, DML, and DCL stand for?
4. What is the difference between CHAR and VARCHAR?
5. What is the difference between primary key and candidate key?
6. What is the difference between DELETE TABLE and TRUNCATE TABLE & DROP table commands in ?

|  |  |
| --- | --- |
| **Assignment No.** | **2** |
| **Title** | **SQL Queries:** |
| **PROBLEM STATEMENT/DEFINITION** | 1. Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym, different constraints etc. 2. Write at least 10 SQL queries on the suitable database application using SQL DML statements |
| **Objectives** | * Understand & implement the various DDL Commands. * Understand database concepts like view, index ,sequence and synonym |
| **Software packages and hardware apparatus used** | MySQL/Oracle  PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse |
| **References** | Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X  Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4  mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | * Date * Title * Problem Definition * Learning Objective * Learning Outcome * Theory-Related concept,Architecture,Syntax etc * Class Diagram/ER diagram * Test cases * Program Listing * Output * Conclusion |

### Assignment No. 2

**Title: SQL Queries**

* + - 1. Design and Develop SQL DDL statements which demonstrate the use of SQL

objects such as Table, View, Index, Sequence, Synonym, different constraints etc.

* + - 1. Write at least 10 SQL queries on the suitable database application using SQL DML statements

**Objectives:** Understand & implement the various DDL Commands.

Understand database concepts like view, index ,sequence and synonym

**Theory:** SQL – Structured Query Language

##### Data Definition in SQL Creating Tables Syntax:-

Create table<table name> (colume\_name 1 datatype size(), colume\_name 2 datatype size(),

….

colume\_name n datatype size());

**e.g.** Create table student with the following fields(name,roll,class,branch)

Create table student

(name char(20), Roll number(5), Class char(10), Branch char(15));

**A table from a table**

* **Syntax :**

##### CREATE TABLE <TableName> (<ColumnName>, <Columnname>) AS SELECT <ColumnName>, <Columnname> FROM <TableName>;

- If the source table contains the records, then new table is also created with the same records present in the source table.

* If you want only structure without records then select statement must have condition. Syntax:

##### CREATE TABLE <TableName> (<ColumnName>, <Columnname>) AS SELECT <ColumnName>, <Columnname> FROM <TableName> WHERE 1=2; (Or)

**CREATE TABLE <TableName> (<ColumnName>, <Columnname>) AS SELECT <ColumnName>, <Columnname> FROM <TableName> WHERE ColumnName =NULL;**

##### Constraints

The definition of a table may include the specification of integrity constraints. Basically two types of constraints are provided: column constraints are associated with a single column whereas table constraints are typically associated with more than one column. A constraint can be named. It is advisable to name a constraint in order to get more meaningful information when this constraint is violated due to, e.g., an insertion of a tuple that violates the constraint. If no name is specified for the constraint, Oracle automatically generates a name of the pattern SYS C<number>.Rules are enforced on data being stored in a table, are called **Constraints**.

Both the Create table & Alter Table SQL can be used to write SQL sentences that attach constraints.

Basically constraints are of three types

1. Domain
   * Not Null
   * Check
2. Entity
   * Primary Key
   * Unique
3. Referential
   * Foreign key
4. Not Null:-Not null constraint can be applied at column level only.

We can define these constraints

1. at the time of table creation Syntax :

**CREATE TABLE <tableName> (<ColumnName> datatype(size) NOT NULL,**

##### <ColumnName> datatype(size),….

**);**

1. After the table creation

**ALTER TABLE <tableName> Modify(<ColumnName> datatype(size) NOT NULL );**

Check constraints

* + Can be bound to column or a table using CREATE TABLE or ALTER TABLE command.
  + Checks are performed when write operation is performed .
  + Insert or update statement causes the relevant check constraint.
  + Ensures the integrity of the data in tables.

Syntax :

* Check constraints at column level Syntax :

##### CREATE TABLE <tableName> (<ColumnName>datatype(size)CHECK(columnName condition),<columnname datatype(size));

**CREATE TABLE <tableName>**

**(<ColumnName> datatype(size) CONSTRAINT <constraint\_name>**

##### CHECK (columnName condition),..

**);**

* Check constraints at table level Syntax :

##### CREATE TABLE <tableName>

**(<ColumnName> datatype(size),**

##### <ColumnName> datatype(size),

**CONSTRAINT <constraint\_name> CHECK (columnName condition),..);**

* Check constraints at table level

Syntax :

**CREATE TABLE**

**<tableName> (<ColumnName> datatype(size),**

**<ColumnName> datatype(size),….,**

##### CHECK (columnName condition)); After table creation

**Alter table tablename**

##### Add constraints constraintname ckeck(condition) The PRIMARY KEY Constraint

A primary key is one or more column(s) in a table used to uniquely identify each row in the table.

* + A table can have only one primary key.
  + Can not be left blank
  + Data must be UNIQUE.
  + Not allows null values
  + Not allows duplicate values.
  + Unique index is created automatically if there is a primary key. Primary key constraint defined at column level

Syntax:

##### CREATE TABLE <TableName>

**(<ColumnName1> <DataType>(<Size>)PRIMARY**

##### KEY,<columnname2

**<datatype(<size>),…..);**

* Primary key constraint defined at Table level Syntax:

##### CREATE TABLE <TableName>

**(<ColumnName1> <DataType>(<Size>) ,…,**

**PRIMARY**

##### KEY(<ColumnName1> <ColumnName2>));

* key constraint defined at Table level

Syntax:

##### CREATE TABLE <TableName>

**(<ColumnName1> <DataType>(<Size>) <columnname2 datatype<(size)<,<columnname3 datatype<size>constraint constraintname PRIMARY KEY(<ColumnName1>));**

After table creation

Alter table tablename

Add(constraint constraintname primary key(columnname));

##### The Unique Key Constraint

* The unique column constraint permits multiple entries of NULL into the column.
* Unique key not allowed duplicate values
* Unique index is automatically created.
* Table can have more than one unique key.
* UNIQUE constraint defined at column level Syntax :

Create table tablename(**<columnname> <datatype>(<Size>**

##### UNIQUE),<columnname> datatype(<size>) );

UNIQUE constraint defined at table level Syntax :

##### CREATE TABLE tablename (<columnname> <datatype>(<Size>),

**<columnname> <datatype>(<Size>), UNIQUE(<columnname>,**

**<columnname> ));**

**After table creation**

##### Alter table tablename

**Add constraint constraintname unique(columnname);**

 The Foreign Key (Self Reference) Constraint Foreign key represents relationships between tables.

A foreign key is a column( or group of columns) whose values are derived from primary key or unique key of some other table.

Foreign key constraint defined at column level Syntax:

##### <columnName> <DataType> (<size>) REFERENCES <TableName>[(<ColumnName>)] [ON DELETE CASCADE]

* + If the ON DELETE CASCADE option is set, a DELETE operation in the master table will trigger a DELETE operation for corresponding records in all detail tables.
  + If the ON DELETE SET NULL option is set, a DELETE operation in the master table will set the value held by the foreign key of the detail tables to null.

Foreign key :

##### ALTER TABLE <child\_tablename> ADD CONSTRAINT <constraint\_name> FOREIGN KEY (<columnname in child\_table>) REFERENCES <parent table name>;

1. FOREIGN KEY constraint at table level
2. FOREIGN KEY constraint defined with ON DELETE CASCADE FOREIGN KEY(<ColumnName>[,<columnname>]) REFERENCES

<TableName> [(<ColumnName>, <ColumnName>) ON DELETE

CASCADE

* + FOREIGN KEY constraint defined with ON DELETE SET NULL

##### FOREIGN KEY(<ColumnName>[,<columnname>]) REFERENCES

**<TableName> [(<ColumnName>, <ColumnName>) ON DELETE SET NULL**

* + To view the constraint Syntax:

##### Select constraint\_name, constraint\_type, search\_condition from user\_constraints where table\_name=<tablename>;

**Select constraint\_name, column\_name from user\_cons\_columns where table\_name=<tablename>;**

To drop the constraints Syntax:-

##### Drop constraint constraintname; Describe commands

To view the structure of the table created use the **DESCRIBE** command.The command displays the column names and datatypes

Syntax:-

Desc[ribe]<table\_name>

e.g desc student

##### Restrictions for creating a table:

1.Table names and column names must begin with a letter. 2.Table names and column names can be 1 to 30 characters long.

3.table names must contain only the characters A-Z,a-z,0-9,underscore\_,$ and # 4.Table name should not be same as the name of another database object.

1. Table name must not be an ORACLE reserved word.
2. Column names should not be duplicate within a table definition.

##### Alteration of TABLE:- Alter table command Syntax:-

Case1:-

Alter table <table\_name>

Add( colume\_name 1 datatype size(), colume\_name 2 datatype size(),

colume\_name n datatype size());

Case2:-

Alter table <table\_name> Modify(colume\_name 1 datatype size(),

colume\_name 2 datatype size(),

…...

colume\_name n datatype size());

After you create a table, you may need to change the table structures because you need to have a column definition needs to be changed. Alter table statement can be used for this purpose.

You can add columns to a table using the alter table statement with the ADD clause.

E.g. Suppose you want to add enroll\_no in the student table then we write

##### Alter table student Add(enroll\_no number(10));

You can modify existing column in a table by using the alter table statement with modify clause.

E.g. Suppose you want to modify or chang the size of previously defined field name in the student table then we write

##### Alter table student modify(name char(25)); Dropping a column from a table

Syntax :

##### ALTER TABLE <Tablename> DROP COLUMN <ColumnName> ;

**Drop table command Syntax:-**

Drop table <table\_name>

Drop table command remnoves the definitions of an oracle table.When you drop a table

,the database loses all the data in the table and all the indexes associated with it.

e.g drop table student;

##### Truncate table command Syntax:-

Trunc table<table\_name>

The truncate table statement is used to remove all rows from a table and to release the storage space used by the table.

e.g.Trunc table student;

##### Rename table command Syntax:-

Rename<oldtable\_name> to<newtable\_name>

Rename statement is used to rename a table,view,sequence,or synonym. e.g. Rename student to stud;

##### Database objects:- Index

An index is a schema object that can speed up retrieval of rows by using pointer. An index provides direct & fast access to rows in a table. Index can be created explicitly or automatically.

Automatically :- A unique index is created automatically when you define a primary key or unique key constraint in a table definition.

Manually :- users can create non unique indexes or columns to speed up access time to the rows.

##### Syntax:

Create index<index\_name> On table(column[ , column]…); Eg. Create index emp\_ename\_idx On emp(ename);

When to create an index

1. The column is used frequently in the WHERE clause or in a join condition.
2. The column contains a wide range of values.
3. The column contains a large number of values.

To display created index of a table eg.

Select ic.index\_name, ic.column\_name, ic.colun\_position col\_pos, ix.uniqueness from

user\_indexes ix, user\_ind\_columns ic where ic.index\_name=ix.index\_name and ic.table\_name=‟emp‟;

##### Removing an Index

Syntax:-

Drop index <index\_name>; eg. Drop index emp\_name\_idx;

**Note:** 1) we cannot modify indexes.

2) To change an index, we must drop it and the re-create it.

##### Views

View is a logical representation of subsets of data from one or more tables. A view takes the output of a query and treats it as a table therefore view can be called as stored query or a virtual table.The tables upon which a view is based are called base tables. In Oracle the SQL command to create a view (virtual table) has the form

##### Create [or replace] view <view-name> [(<column(s)>)] as

**<select-statement> [with check option [constraint <name>]];**

The optional clause or replace re-creates the view if it already exists. <column(s)> names the columns of the view. If <column(s)> is not specified in the view definition, the columns of the view get the same names as the attributes listed in the select statement (if possible).

Example: The following view contains the name, job title and the annual salary of employees working in the department 20:

##### Create view DEPT20 as

**select ENAME, JOB, SAL 12 ANNUAL SALARY from EMP where DEPTNO = 20;**

In the select statement the column alias ANNUAL SALARY is specified for the expression SAL∗12 and this alias is taken by the view. An alternative formulation of the above view

definition is

##### Create view DEPT20 (ENAME, JOB, ANNUAL SALARY) as select ENAME, JOB, SAL

**from EMP where DEPTNO = 20;**

A view can be used in the same way as a table, that is, rows can be retrieved from a view(also respective rows are not physically stored, but derived on basis of the select statement inthe view definition), or rows can even be modified. A view is evaluated again each time it is accessed. In Oracle SQL no insert, update, or delete modifications on views are allowed

that use one of the following constructs in the view definition:

* Joins
* Aggregate function such as sum, min, max etc.
* set-valued subqueries (in, any, all) or test for existence (exists)
* group by clause or distinct clause

In combination with the clause with check option any update or insertion of a row into the view is rejected if the new/modified row does not meet the view definition, i.e., these rows would not be selected based on the select statement. A with check option can be named using the constraint clause.

A view can be deleted using the command delete <view-name>. To describe the structure of a view

e.g. **Describe stud;**

To display the contents of view e.g. Select \* from stud

##### Removing a view:

Syntax:- **Drop view <view\_name>**

e.g. Drop view stud

##### Sequence:

A sequence is a database object, which can generate unique, sequential integer values. It can be used to automatically generate primary key or unique key values. A sequence can be either in an ascending or descending order.

Syntax :

Create

sequence<sequence\_name> [increment by n]

[start with n]

[{maxvalue n | nomaxvalue}] [{minvalue n| nominvalue}] [{cycle | nocycle}]

[{cache n| nocache}];

|  |  |
| --- | --- |
| Increment by n | Specifies the interval between sequence number where n is an integer. If this clause is omitted, the  sequence is increment by 1. |
| Start with n | Specifies the first sequence number to be generated. If this clause is omitted ,  the sequence is start with 1. |
| Maxvalue n | Specifies the maximum value, the  sequence can generate |
| Nomax value n | Specifies the maximum value of 10e27-1 for an ascending sequence & - 1 for descending sequence. This is a  default option. |
| Minvalue n | Specifies the minimum sequence  value. |
| Nominvalue n | Specifies the minimum value of 1 for an ascending & 10e26-1 for descending sequence. This is a default  option. |
| Cycle | Specifies that the sequence continues  to generate values from the beginning after reaching either its max or min value. |
| Nocycle | Specifies that the sequence can not generate more values after reaching either its max or min value. This is a  default option. |
| Cache / nocache | Specifies how many values the oracle server will preallocate & keep in memory. By default, the oracle server  will cache 20 values. |

After creating a sequence we can access its values with the help of pseudo columns like **curval** & **nextval**.

**Nextval :** nextval returns initial value of the sequence when reference to for the first time. Last references to the nextval will increment the sequence using the increment by clause & returns the new value.

**Curval** : curval returns the current value of the sequence which is the value returned by the last reference to last value.

##### Modifyning a sequence:

The sequence can be modified when we want to perform the following :

## □ 

 Set or eliminate minvalue or maxvalue

□ 

 Change the increment value.



Change the number of cache sequence number.

 Syntax :

Alter sequence <sequence\_name>

 [increment by n]

## 

 [start with n]

[{maxvalue n | nomaxvalue}]

[{minvalue n| nominvalue}]

## 

 [{cycle | nocycle}] 

[{cache n| nocache}];

##### Synonym:

A synonym is a database object,which is used as an alias(alternative name)for a table,view or sequence.

Syntax:-

In the syntax

Create[public]synonym

<synonym\_name>for<table\_name>;

Public:-Creates a synonym accessible to all users. Synonym:-Is the name of the synonym to be created.

Synonym can either be private or public.A private synonym is created by normal user,which is available to that persons.

A public synonym is created by a database administrator(DBA),which can be availed by any other database user.

##### Uses:-

* 1. Simplify SQL statements.
  2. Hide the name and owner of an object. 3.Provide public access to an object.

Guidelines:-

1.User can do all DML manipulations such as insert ,delete,update on synonym. 2.User cannot perform any DDL operations on the synonym except dropping the synonym.

3.All the manipulations on it actually affect the table e.g Create synonym stud1 for student;

SQL, pronounced SEQUEL, is the standard language to access relational databases. SQL is an abbreviation for Structured Query Language. I'll just add that SQL is composed of DML and DDL. DML are the keywords you use to access and manipulate data, hence the name Data Manipulation Language. DDL are the keywords you use to create objects such as views, tables and procedures, hence the name Data Definition Language.

##### Tables

In relational database systems (DBS) data are represented using tables (relations). A query issued against the DBS also results in a table. A table has the following structure:

|  |  |  |  |
| --- | --- | --- | --- |
| Column 1 | Column 2 | . . . | Column n |
| ← Tuple (or | Record) |  |  |
|  |  |  |  |
| ………. | ………. | ………. | ……….. |

Attributes

A table is uniquely identiﬁed by its name and consists of rows that contain the stored information, each row containing exactly one tuple (or record). A table can have one or more columns.

A column is made up of a column name and a data type, and it describes an attribute of the tuples. The structure of a table, also called relation schema, thus is deﬁned by its attributes.The type of information to be stored in a table is deﬁned by the data types of the attributes at table

creation time.SQL uses the terms table, row, and column for relation, tuple, and attribute, respectively.

A table can have up to 254 columns which may have different event or same data types and sets of

values (domains), respectively. Possible domains are alphanumeric data (strings), numbers and date formats.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Datatype | **Description** | **Max Size: Oracle 7** | **Max Size: Oracle 8** | **Max Size: Oracle 9** | **Max Size: PL/SQL** | **PL/SQL**  **Subtypes/ Synonyms** |
| VARCHAR2(s  ize) | Variable length  character string having maximum length *size* bytes.  You must specify size | 2000  bytes minimum is 1 | **4000**  bytes minimu m is 1 | **4000** bytes  minimum is 1 | 32767  bytes minimum is 1 | STRING VARCHA R |
| NVARCHAR2  (size) | Variable length  national character set string having maximum length *size* bytes.  You must specify size | N/A | 4000  bytes minimu m is 1 | 4000 bytes  minimum is 1 | 32767  bytes minimum is 1 | STRING VARCHA R |
| VARCHAR | Now deprecated  - VARCHAR is  a synonym for VARCHAR2  but this usage may change in  future versions. | - | - | - |  |  |
| CHAR(size) | Fixed length character data of length size bytes. This should be used for fixed length data. Such as codes A100, B102... | 255 bytes Default and minimum size is 1 byte. | **2000**  bytes Default and minimu m size is 1 byte. | **2000** bytes Default and minimum size is 1 byte. | 32767  bytes Default and minimum size is 1 byte. | CHARAC TER |
| NCHAR  (size) | Fixed length  national character set | N/A | 2000  bytes Default | 2000 bytes  Default and minimum size | 32767  bytes Default |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | data of length size bytes. This should be used for fixed length data. Such as codes A100, B102... |  | and minimu m size is 1 byte. | is 1 byte. | and minimum size is 1 byte. |  |
| NUMBER(p,s) | Number having precision p and scale s. | The precision p can range from 1 to  38.  The scale s can range from -84  to 127. | The precisio n p can range from 1  to 38.  The scale s can range from -84  to 127. | The precision p can range from 1 to 38.  The scale s can range from -84 to  127. | Magnitud  e  1E-130 ..  10E125  maximum precision of 126 binary digits, which is roughly equivalen t to 38 decimal digits  The scale s can range from -84  to 127. For floating point don't specify p,s REAL  has a maximum precision of 63 binary digits, which is roughly equivalen t to 18 decimal digits | fixed-point numbers: DEC DECIMAL NUMERIC  floating- point: DOUBLE PRECISIO N FLOAT  binary\_dou ble binary\_floa t  integers: INTEGER INT SMALLIN T  simple\_inte ger(10g)  BOOLEA N  REAL |
| PLS\_INTEGE R | signed integers  PLS\_INTEGER  values require less storage and | PL/SQL  only | PL/SQL  only | PL/SQL only | magnitud  e range is  - 21474836 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | provide better performance than NUMBER values.  So use PLS\_INTEGER  where you can! |  |  |  | 47 ..  21474836  47 |  |
| BINARY\_INT EGER | signed integers (older slower version of PLS\_INTEGER  ) |  |  |  | magnitud e range is  - 21474836  47 ..  21474836  47 | NATURA L NATURA LN POSITIVE POSITIVE N SIGNTYP  E |
| LONG | Character data of variable length (A bigger version the VARCHAR2  datatype) | 2  Gigabytes | 2  Gigabyt es | 2 Gigabytes - but now deprecated | 32760  bytes Note this is smalller than the maximum width of a LONG  column |  |
| DATE | Valid date range | from January 1,  4712 BC  to December 31, 4712 AD. | from January 1, 4712  BC to Decemb er 31,  **9999**  AD. | from January 1, 4712 BC to  December 31,  **9999** AD. | from January 1,  4712 BC  to December 31, **9999** AD.  (in Oracle7 = 4712 AD) |  |
| TIMESTAMP  (fractional\_sec onds\_precision  ) | the number of digits in the fractional part of the SECOND datetime field. | - | - | Accepted  values of fractional\_sec onds\_precisio n are 0 to 9.  (default = 6) |  |  |
| TIMESTAMP  (fractional\_sec onds\_precision  ) WITH  {LOCAL} TIMEZONE | As above with time zone displacement value | - | - | Accepted values of fractional\_sec onds\_precisio n are 0 to 9.  (default = 6) |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| YEAR  (year\_precision  ) TO MONTH | and months, where year\_precision is the number of digits in the YEAR datetime field. |  |  | values are 0 to  9. (default = 2) |  |  |
| INTERVAL DAY  (day\_precision) TO SECOND  (fractional\_sec onds\_precision  ) | Time in days,  hours, minutes, and seconds.  *day\_precision* is the maximum number of digits in 'DAY'  *fractional\_secon ds\_precision* is the max number of fractional digits in the SECOND field. | - | - | *day\_precision*  may be 0 to 9.  (default = 2)  *fractional\_sec onds\_precisio n* may be 0 to  9. (default = 6) |  |  |
| RAW(size) | Raw binary data  of length size bytes.  You must specify size for a RAW value. | Maximum size is 255 bytes. | Maximu m size is **2000**  bytes | Maximum size is **2000** bytes | 32767  bytes |  |
| LONG RAW | Raw binary data of variable length. (not intrepreted by PL/SQL) | 2  Gigabytes  . | 2  Gigabyt es. | 2 Gigabytes - but now deprecated | 32760  bytes Note this is smalller than the maximum width of a LONG RAW  column |  |
| ROWID | Hexadecimal string representing the unique address of a row in its table. (primarily for values returned by the ROWID pseudocolumn.) | 8 bytes | 10 bytes | 10 bytes | Hexadeci mal string representi ng the unique address of a row in its table. (primarily for values  returned |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | by the ROWID  pseudocol umn.) |  |
| UROWID | Hex string representing the logical address of a row of an index-organized table | N/A | The maximu m size and default is 4000 bytes | The maximum size and default is 4000 bytes | universal  rowid - Hex string representi ng the logical address of a row of an index- organized table, either physical, logical, or foreign (non-  Oracle) | See CHARTO ROWID  and the package: DBMS\_R OWID |
| MLSLABEL | Binary format of an operating system label.This datatype is used with Trusted Oracle7. |  |  |  |  |  |
| CLOB | Character Large  Object | 4Gigabyte  s | 4Gigaby  tes | 4Gigabytes | 4Gigabyt  es |  |
| NCLOB | National Character Large Object |  | 4Gigaby tes | 4Gigabytes | 4Gigabyt es |  |
| BLOB | Binary Large Object |  | 4Gigaby tes | 4Gigabytes | 4Gigabyt es |  |
| BFILE | pointer to binary file on disk |  | 4Gigaby tes | 4Gigabytes | The size  of a BFILE is  system dependent but cannot exceed four gigabytes (2\*\*32 -  1 bytes). |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| XMLType | XML data | - | - | 4Gigabytes | Populate with XML  from a CLOB or VARCH AR2.  or query from another XMLTyp  e column. |  |

**FAQ :** Consider relational schema **Student( Roll\_no, Name, Deptno, Marks,Email\_id )**

Develop SQL DDL statements.

1. Create table Student;
2. Insert values in student table.
3. Add a new attribute date of birth in student record using alter statement.
4. Drop date of birth attribute from student table.
5. Update a student marks where roll no is 7;
6. Delete a record of student whose roll no is 4;
7. Create view for student table;
8. Create index on Roll no in student table.
9. Create sequence on student table.
10. Create synonym on student table.

##### □ Fetching Data from Table:

The SQL **SELECT** command is used to fetch data from database. You can use this command at

> prompt as well as in any script like PHP. Syntax:

Here is generic SQL syntax of SELECT command to fetch data from table:

SELECT field1, field2,...fieldN table\_name1, table\_name2... [WHERE Clause]

[OFFSET M ][LIMIT N]

* You can use one or more tables separated by comma to include various conditions using a WHERE clause, but WHERE clause is an optional part of SELECT command.
* You can fetch one or more fields in a single SELECT command.
* You can specify star (\*) in place of fields. In this case, SELECT will return all the fields.
* You can specify any condition using WHERE clause.
* You can specify an offset using **OFFSET** from where SELECT will start returning records. By default offset is zero.
* You can limit the number of returns using **LIMIT** attribute.

##### Fetching Data from Command Prompt:

This will use SQL SELECT command to fetch data from table tutorials\_tbl Example:

Following example will return all the records from **tutorials\_tbl** table:

> SELECT \* from tutorials\_tbl

+++++

| tutorial\_id | tutorial\_title | tutorial\_author | submission\_date |

+++++

|

|

|

1 | Learn PHP

| John Poul

| 2007-05-21|

2 | Learn

| Abdul S

| 2007-05-21

|

3 | JAVA Tutorial | Sanjay

| 2007-05-21

|

+++++

The SQL **SELECT** statement returns a result set of records from one or more tables. A **SELECT** statement retrieves zero or more rows from one or more database tables or database views. In most applications, SELECT is the most commonly used Data Manipulation Language (DML) command. As SQL is a declarative programming language, SELECT queries specify a result set, but do not specify how to calculate it. The database translates the query into a "query plan" which may vary between executions, database versions and database software. This functionality is called the "query optimizer" as it is responsible for finding the best possible execution plan for the query, within applicable constraints.

The SELECT statement has many optional clauses:

* WHERE specifies which rows to retrieve.
* GROUP BY groups rows sharing a property so that an aggregate function can be applied to each group.
* HAVING selects among the groups defined by the GROUP BY clause.
* ORDER BY specifies an order in which to return the rows.
* AS provides an alias which can be used to temporarily rename tables or columns.

##### WHERE Clause

We have seen SQL **SELECT** command to fetch data from table. We can use a conditional clause called **WHERE** clause to filter out results. Using WHERE clause, we can specify a selection criteria to select required records from a table.

##### Syntax:

Here is generic SQL syntax of SELECT command with WHERE clause to fetch data from table:

SELECT field1, field2,...fieldN table\_name1, table\_name2... [WHERE condition1 [AND [OR]] condition2.....

* You can use one or more tables separated by comma to include various conditions using a WHERE clause, but WHERE clause is an optional part of SELECT command.
* You can specify any condition using WHERE clause.
* You can specify more than one conditions using **AND** or **OR** operators.
* A WHERE clause can be used along with DELETE or UPDATE SQL command also to specify a condition.

The **WHERE** clause works like an if condition in any programming language. This clause is used to compare given value with the field value available in table. If given value from outside is equal to the available field value in table, then it returns that row.

Here is the list of operators, which can be used with **WHERE** clause. Assume field A holds 10 and field B holds 20, then:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
|  | Checks if the values of two operands are equal or not, if yes then condition becomes true. |  |
| = | (A = B) is not true. |
|  |  |
|  | Checks if the values of two operands are equal or not, if values are  not equal then condition becomes true. |  |
| != | (A != B) is true. |
|  |  |
|  | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. |  |
| > | (A > B) is not true. |
|  |  |
|  | Checks if the value of left operand is less than the value of right  operand, if yes then condition becomes true. |  |
| < | (A < B) is true. |
|  |  |
|  | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| >= |
|  |
|  | Checks if the value of left operand is less than or equal to the value  of right operand, if yes then condition becomes true. |  |
| <= | (A <= B) is true. |
|  |  |

The WHERE clause is very useful when you want to fetch selected rows from a table, especially when you use **Join**.

It is a common practice to search records using **Primary Key** to make search fast.

If given condition does not match any record in the table, then query would not return any row.

##### Fetching Data from Command Prompt:

This will use SQL SELECT command with WHERE clause to fetch selected data from table tutorials\_tbl.

##### Example:

Following example will return all the records from **tutorials\_tbl** table for which author name is **Sanjay**:

> SELECT \* from tutorials\_tbl WHERE tutorial\_author='Sanjay';

+ + + + +

| tutorial\_id | tutorial\_title | tutorial\_author | submission\_date |

+ + + + +

| 3 | JAVA Tutorial | Sanjay | 2007-05-21 |

+ + + + +

Unless performing a **LIKE** comparison on a string, the comparison is not case sensitive. You can make your search case sensitive using **BINARY** keyword as follows:

SELECT \* from tutorials\_tbl WHERE BINARY tutorial\_author='sanjay';

##### LIKE Clause

We have seen SQL **SELECT** command to fetch data from table. We can also use a conditional clause called **WHERE** clause to select required records.

A WHERE clause with equals sign (=) works fine where we want to do an exact match. Like if "tutorial\_author = 'Sanjay'". But there may be a requirement where we want to filter out all the results where tutorial\_author name should contain "jay". This can be handled using SQL **LIKE** clause along with WHERE clause.

If SQL LIKE clause is used along with % characters, then it will work like a meta character (\*) in UNIX while listing out all the files or directories at command prompt.

Without a % character, LIKE clause is very similar to equals sign along with WHERE clause.

##### Syntax:

Here is generic SQL syntax of SELECT command along with LIKE clause to fetch data from table:

SELECT field1, field2,...fieldN table\_name1, table\_name2... WHERE field1 LIKE condition1 [AND [OR]] filed2 = 'somevalue'

* You can specify any condition using WHERE clause.
* You can use LIKE clause along with WHERE clause.
* You can use LIKE clause in place of equals sign.
* When LIKE is used along with % sign then it will work like a meta character search.
* You can specify more than one conditions using **AND** or **OR** operators.
* A WHERE...LIKE clause can be used along with DELETE or UPDATE SQL command also to specify a condition.

##### Using LIKE clause at Command Prompt:

This will use SQL SELECT command with WHERE...LIKE clause to fetch selected data from table tutorials\_tbl.

##### Example:

Following example will return all the records from **tutorials\_tbl** table for which author name ends with **jay**:

SELECT \* from tutorials\_tbl WHERE tutorial\_author LIKE '%jay';

+ + + + +

| tutorial\_id | tutorial\_title | tutorial\_author | submission\_date |

+ + + + +

| 3 | JAVA Tutorial | Sanjay | 2007-05-21 |

+ + + + +

##### GROUP BY Clause

You can use **GROUP BY** to group values from a column, and, if you wish, perform calculations on that column. You can use COUNT, SUM, AVG, etc., functions on the grouped column.

To understand **GROUP BY** clause, consider an **employee\_tbl** table, which is having the following records:

> SELECT \* FROM employee\_tbl;

+ + + + +

| id | name | work\_date | daily\_typing\_pages |

+ + + + +

| 1 | John | 2007-01-24 | 250 |

| 2 | Ram | 2007-05-27 | 220 |

| 3 | Jack | 2007-05-06 | 170 |

| 3 | Jack | 2007-04-06 | 100 |

| 4 | Jill | 2007-04-06 | 220 |

| 5 | Zara | 2007-06-06 | 300 |

| 5 | Zara | 2007-02-06 | 350 |

+ + + + + 7 rows in set (0.00 sec)

Now, suppose based on the above table we want to count number of days each employee did work.

If we will write a SQL query as follows, then we will get the following result:

SELECT COUNT(\*) FROM employee\_tbl;

+ +

| COUNT(\*) |

+ +

| 7 |

But this is not serving our purpose, we want to display total number of pages typed by each person separately. This is done by using aggregate functions in conjunction with a **GROUP BY** clause as follows:

SELECT name, COUNT(\*) FROM employee\_tbl **GROUP BY** name;

+++

| name | COUNT(\*) |

+++

| Jack |2 |

| Jill |1 |

| John |1 |

| Ram |1 |

| Zara |2 |

+++

5 rows in set (0.04 sec)

We will see more functionality related to GROUP BY in other functions like SUM, AVG, etc.

##### COUNT Function

**COUNT** Function is the simplest function and very useful in counting the number of records, which are expected to be returned by a SELECT statement.

To understand **COUNT** function, consider an **employee\_tbl** table, which is having the following records:

> SELECT \* FROM employee\_tbl;

+ + + + +

| id | name | work\_date | daily\_typing\_pages |

+ + + + +

| 1 | John | 2007-01-24 | 250 |

| 2 | Ram | 2007-05-27 | 220 |

| 3 | Jack | 2007-05-06 | 170 |

| 3 | Jack | 2007-04-06 | 100 |

| 4 | Jill | 2007-04-06 | 220 |

| 5 | Zara | 2007-06-06 | 300 |

| 5 | Zara | 2007-02-06 | 350 |

+ + + + + 7 rows in set (0.00 sec)

Now, suppose based on the above table you want to count total number of rows in this table, then you can do it as follows:

>SELECT COUNT(\*) FROM employee\_tbl ;

+ +

| COUNT(\*) |

+ +

| 7 |

+ +

1 row in set (0.01 sec)

Similarly, if you want to count the number of records for Zara, then it can be done as follows:

| COUNT(\*) |

+ +

| 2 |

+ +

1 row in set (0.04 sec)

**NOTE:** All the SQL queries are case insensitive so it does not make any difference if you give ZARA or Zara in WHERE condition.

##### MAX Function

**MAX** function is used to find out the record with maximum value among a record set.

To understand **MAX** function, consider an **employee\_tbl** table, which is having the following records:

> SELECT \* FROM employee\_tbl;

+ + + + +

| id | name | work\_date | daily\_typing\_pages |

+ + + + +

| 1 | John | 2007-01-24 | 250 |

| 2 | Ram | 2007-05-27 | 220 |

| 3 | Jack | 2007-05-06 | 170 |

| 3 | Jack | 2007-04-06 | 100 |

| 4 | Jill | 2007-04-06 | 220 |

| 5 | Zara | 2007-06-06 | 300 |

| 5 | Zara | 2007-02-06 | 350 |

+ + + + + 7 rows in set (0.00 sec)

Now, suppose based on the above table you want to fetch maximum value of daily\_typing\_pages, then you can do so simply using the following command:

SELECT MAX(daily\_typing\_pages) FROM employee\_tbl;

+ +

| MAX(daily\_typing\_pages) |

+ +

| 350 |

+ + 1 row in set (0.00 sec)

You can find all the records with maximum value for each name using **GROUP BY** clause as follows:

|1 | John |

|2 | Ram |

|5 | Zara |

250 |

220 |

350 |

++++ 5 rows in set (0.00 sec)

You can use **MIN** Function along with **MAX** function to find out minimum value as well. Try out the following example:

SELECT MIN(daily\_typing\_pages) least, MAX(daily\_typing\_pages) max FROM employee\_tbl;

+ + +

| least | max |

+ + +

| 100 | 350 |

+ + +

1 row in set (0.01 sec)

##### MIN Function

**MIN** function is used to find out the record with minimum value among a record set.

To understand **MIN** function, consider an **employee\_tbl** table, which is having the following records:

SELECT \* FROM employee\_tbl;

+ + + + +

| id | name | work\_date | daily\_typing\_pages |

+ + + + +

| 1 | John | 2007-01-24 | 250 |

| 2 | Ram | 2007-05-27 | 220 |

| 3 | Jack | 2007-05-06 | 170 |

| 3 | Jack | 2007-04-06 | 100 |

| 4 | Jill | 2007-04-06 | 220 |

| 5 | Zara | 2007-06-06 | 300 |

| 5 | Zara | 2007-02-06 | 350 |

+ + + + + 7 rows in set (0.00 sec)

Now, suppose based on the above table you want to fetch minimum value of daily\_typing\_pages, then you can do so simply using the following command:

You can find all the records with minimum value for each name using **GROUP BY** clause as follows:

SELECT id, name, MIN(daily\_typing\_pages) FROM employee\_tbl GROUP BY name;

+ + + +

| id | name | MIN(daily\_typing\_pages) |

+ + + +

| 3 | Jack | 100 |

| 4 | Jill | 220 |

| 1 | John | 250 |

| 2 | Ram | 220 |

| 5 | Zara | 300 |

+ + + + 5 rows in set (0.00 sec)

You can use **MIN** Function along with **MAX** function to find out minimum value as well. Try out the following example:

SELECT MIN(daily\_typing\_pages) least, MAX(daily\_typing\_pages) max FROM employee\_tbl;

+ + +

| least | max |

+ + +

| 100 | 350 |

+ + +

1 row in set (0.01 sec)

##### AVG Function

**AVG** function is used to find out the average of a field in various records.

To understand **AVG** function, consider an **employee\_tbl** table, which is having following records:

SELECT \* FROM employee\_tbl;

+ + + + +

| id | name | work\_date | daily\_typing\_pages |

+ + + + +

| 1 | John | 2007-01-24 | 250 |

| 2 | Ram | 2007-05-27 | 220 |

| 3 | Jack | 2007-05-06 | 170 |

| 3 | Jack | 2007-04-06 | 100 |

| 4 | Jill | 2007-04-06 | 220 |

| 5 | Zara | 2007-06-06 | 300 |

| 5 | Zara | 2007-02-06 | 350 |

+ + + + + 7 rows in set (0.00 sec)

Now, suppose based on the above table you want to calculate average of all the dialy\_typing\_pages, then you can do so by using the following command:

SELECT AVG(daily\_typing\_pages) FROM employee\_tbl;

+ +

| AVG(daily\_typing\_pages) |

+ +

| 230.0000 |

+ + 1 row in set (0.03 sec)

You can take average of various records set using **GROUP BY** clause. Following example will take average all the records related to a single person and you will have average typed pages by every person.

SELECT name, AVG(daily\_typing\_pages) FROM employee\_tbl GROUP BY name;

+ + +

| name | AVG(daily\_typing\_pages) |

+ + +

| Jack | 135.0000 |

| Jill | 220.0000 |

| John | 250.0000 |

| Ram | 220.0000 |

| Zara | 325.0000 |

+ + + 5 rows in set (0.20 sec)

##### SUM Function

**SUM** function is used to find out the sum of a field in various records.

To understand **SUM** function, consider an **employee\_tbl** table, which is having the following records:

SELECT \* FROM employee\_tbl;

+ + + + +

| id | name | work\_date | daily\_typing\_pages |

+ + + + +

| 1 | John | 2007-01-24 | 250 |

| 2 | Ram | 2007-05-27 | 220 |

| 3 | Jack | 2007-05-06 | 170 |

| 3 | Jack | 2007-04-06 | 100 |

| 4 | Jill | 2007-04-06 | 220 |

| 5 | Zara | 2007-06-06 | 300 |

| 5 | Zara | 2007-02-06 | 350 |

+ + + + + 7 rows in set (0.00 sec)

Now, suppose based on the above table you want to calculate total of all the dialy\_typing\_pages, then you can do so by using the following command:

+ +

| 1610 |

+ + 1 row in set (0.00 sec)

You can take sum of various records set using **GROUP BY** clause. Following example will sum up all the records related to a single person and you will have total typed pages by every person.

SELECT name, SUM(daily\_typing\_pages) FROM employee\_tbl GROUP BY name;

+ + +

| name | SUM(daily\_typing\_pages) |

+ + +

| Jack | 270 |

| Jill | 220 |

| John | 250 |

| Ram | 220 |

| Zara | 650 |

+ + + 5 rows in set (0.17 sec)

##### HAVING clause

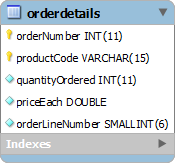
The HAVING clause is used in the SELECT statement to specify filter conditions for group of rows or aggregates. The HAVING clause is often used with the GROUP BY clause. When using with the GROUP BY clause, you can apply a filter condition to the columns that appear in the GROUP BY clause. If the GROUP BY clause is omitted, the HAVING clause behaves like the WHERE clause. Notice that the HAVING clause applies the condition to each group of rows, while the WHERE clause applies the condition to each individual row.

**Examples of using HAVING clause**

Let‟s take a look at an example of using

HAVING clause.

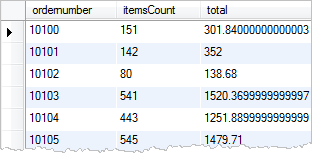
We will use the orderdetails table in the sample database for the sake of demonstration.



We can use the GROUP BY clause to get order number, the number of items sold per order and total sales for each:

SELECT ordernumber, SUM(quantityOrdered) AS itemsCount, SUM(priceeach) AS total

FROM orderdetails GROUP BY ordernumber

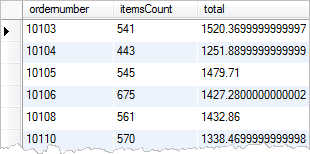


Now, we can find which order has total sales greater than $1000. We use the HAVING clause on the aggregate as follows:

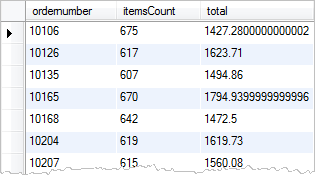
SELECT ordernumber, SUM(quantityOrdered) AS itemsCount, SUM(priceeach) AS total

FROM orderdetails GROUP BY ordernumber

**HAVING** total > 1000



We can construct a complex condition in the HAVING clause using logical operators such as OR and AND. Suppose we want to find which order has total sales greater than $1000 and contains more than 600 items, we can use the following query:



SELECT ordernumber, sum(quantityOrdered) AS itemsCount, sum(priceeach) AS total

FROM orderdetails GROUP BY ordernumber

HAVING total > 1000 AND itemsCount > 600

The HAVING clause is only useful when we use it with the GROUP BY clause to generate the output of the high-level reports. For example, we can use the HAVING clause to answer some kinds of queries like give me all the orders in this month, this quarter and this year that have total sales greater than 10K.

##### UPDATE Query

There may be a requirement where existing data in a table needs to be modified. You can do so by using SQL **UPDATE** command. This will modify any field value of any table.

Syntax:

Here is generic SQL syntax of UPDATE command to modify data into table:

UPDATE table\_name SET field1=new-value1, field2=new-value2 [WHERE Clause]

* You can update one or more field altogether.
* You can specify any condition using WHERE clause.
* You can update values in a single table at a time.

The WHERE clause is very useful when you want to update selected rows in a table. Updating Data from Command Prompt:

This will use SQL UPDATE command with WHERE clause to update selected data into table tutorials\_tbl.

##### Example:

Following example will update **tutorial\_title** field for a record having tutorial\_id as 3.

UPDATE tutorials\_tbl

SET tutorial\_title='Learning JAVA' WHERE tutorial\_id=3;

##### DELETE Query

If you want to delete a record from any table, then you can use SQL command **DELETE FROM**. You can use this command at > prompt as well as in any script like PHP. **Syntax:**

Here is generic SQL syntax of DELETE command to delete data from a table:

DELETE FROM table\_name [WHERE Clause]

* If WHERE clause is not specified, then all the records will be deleted from the given table.
* You can specify any condition using WHERE clause.
* You can delete records in a single table at a time.

The WHERE clause is very useful when you want to delete selected rows in a table. Deleting Data from Command Prompt:

This will use SQL DELETE command with WHERE clause to delete selected data into table tutorials\_tbl.

Example:

Following example will delete a record into tutorial\_tbl whose tutorial\_id is 3.

DELETE FROM tutorials\_tbl WHERE tutorial\_id=3;

Create table **location**(location\_id numeric(3) primary key,regional\_group varchar(15));

Create table **department**(Department\_ID numeric(2) primary key,name varchar(20),location\_id int, foreign key(location\_id) references location(location\_id));

Create table **job**(job\_ID numeric(3) primary key,function varchar(20));

Create table **employee**(employee\_ID numeric(4) primary key,last\_name varchar(20),first\_name varchar(20),middle\_name varchar(20),job\_id numeric(3),manager\_id varchar(20), hired\_date date, salary numeric(6), comm numeric(4), department\_id numeric(2) not null,FOREIGN KEY (job\_id) REFERENCES job(job\_id),FOREIGN KEY (department\_id) REFERENCES department(department\_id));

* 1. List the details about “SMITH”

*Select \* from employee where last\_name=‘SMITH’;*

* 1. List out the employees who are working in department 20

*Select \* from employee where department\_id=20*

* 1. List out the employees who are earning salary between 3000 and 4500

*Select \* from employee where salary between 3000 and 4500*

* 1. List out the employees who are working in department 10 or 20

*Select \* from employee where department\_id in (10,20)*

* 1. Find out the employees who are not working in department 10 or 30

*Select last\_name, salary, comm, department\_id from employee where department\_id not in (10,30)*

* 1. List out the employees whose name starts with “S”

*Select \* from employee where last\_name like 'S%';*

* 1. List out the employees whose name start with “S” and end with “H”

*Select \* from employee where last\_name Like 'S%H';*

* 1. List out the employees whose name length is 5 and start with “S”

*Select \* from employee where last\_name like 'S \_';*

* 1. List out the employees who are working in department 10 and draw the salaries more than 3500

*Select \* from employee where department\_id=10 and salary>3500*

* 1. List out the employees who are not receiving commission.

*Select \* from employee where commission is Null*

* 1. List out the employee id, last name in ascending order based on the employee id.

*Select employee\_id, last\_name from employee order by employee\_id*

* 1. List out the employee id, name in descending order based on salary column

*Select employee\_id, last\_name, salary from employee order by salary desc*

* 1. List out the employee details according to their last\_name in ascending order and salaries in descending order

**Conclusion:** Thus we have studied to use & implement various DML queries.

**FAQ :**

1. Explain DML.
2. Explain INSERT command with syntax.
3. Explain DELETE command with syntax.
4. Explain UPDATE command with syntax.
5. Explain SELECT command with syntax.
6. Enlist different comparisons operator. Explain with example.
7. Enlist different Logical operator. Explain with example.
8. Explain Order by clause.
9. Enlist different Aggregation function. Explain with example.

|  |  |
| --- | --- |
| **Assignment No.** | **3** |
| **Title** | **SQL Queries all types of Join, Sub-Query and View:**  Write at least10 SQL queries for suitable database application using SQL DML statements. |
| **PROBLEM STATEMENT/DEFINITION** | Write at least10 SQL queries for suitable database application using SQL DML statements |
| **Objectives** | To understand   * Types of joins. * Subquery and its types. * Complex views |
| **Software packages and hardware apparatus used** | MySQL/Oracle  PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse |
| **References** | Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X  Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4  mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | * Date * Title * Problem Definition * Learning Objective * Learning Outcome * Theory-Related concept,Architecture,Syntax etc * Class Diagram/ER diagram * Test cases * Program Listing * Output * Conclusion |

### Assignment No: 3

**Title:-** Design at least 10 SQL queries for suitable database application using SQL DML statements: all types of Join, Sub-Query and View.

**Objectives:-** a)Types of joins. b) Subquery and its types c) Complex views

##### THEORY: SQL – Join

The ability of relational „join‟ operator is an important feature of relational systems. A join

makes it possible to select data from more than table by means of a single statement. This joining of tables may be done in a many ways.

Types of JOIN

1. Inner
2. Outer(left, right,full)
3. Cross

##### Inner join :

* + Also known as equi join.
  + Statements generally compares two columns from two columns with the equivalence operator =.
  + This type of join can be used in situations where selecting only those rows that have values in common in the columns specified in the ON clause, is required.
* Syntax :

(ANSI style)

SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1> INNER JOIN <tablename2>ON <tablename1>.<columnname> =

<tablename2>.<columnname> WHERE <condition> ORDER BY <columnname1>; (theta style)

SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1>,

<tablename2> WHERE <tablename1>.<columnname> = <tablename2>.<columnname> AND <condition> ORDER BY <columnname1>;

* List the employee details along with branch names to which they belong.

Emp(empno,fname,lname,dept,desig,branchno) Branch(bname,branchno)

Select e.empno,e.fname,e.lname,e.dept, b.bname, e.desig from emp e inner join branch b on b.branchno=e.branchno;

Select e.empno, e.fname, e.lname, e.dept, b.bname, e.desig from emp e, branch b on where b.branchno=e.branchno;

Eg. List the customers along with the account details associated with them. Customer(custno,fname,lname)

Acc\_cust\_dtls(fdno,custno)

Acc\_mstr(accno,branchno,curbal) Branch\_mstr(name,branchno)

* Select c.custno, c.fname, c.lname, a.accno,a.curbal,b.branchno,b.name from customer c inner join acc\_cust\_dtls k on c.custno=k.custno inner join acc\_mstr a on k.fdno=a.accno inner join branch b on b.branchno=a.branchno where c.custno like „C%‟ order by c.custno;
* Select c.custno, c.fname, c.lname, a.accno,a.curbal,b.branchno,b.name from customer c, acc\_cust\_dtls k, acc\_mstr a, branch b where c.custno=k.custno and k.fdno=a.accno and b.branchno=a.branchno and c.custno like „C%‟ order by c.custno;

##### Outer Join

Outer joins are similar to inner joins, but give a little bit more flexibility when selecting data from related tables. This type of joins can be used in situations where it is desired, to select all rows from the table on left( or right, or both) regardless of whether the other table has values in common & ( usually) enter NULL where data is missing.

* Tables

Emp\_mstr(empno,fname,lname,dept) Cntc\_dtls(codeno,cntc\_type,cntc\_data)

##### Left Outer Join

List the employee details along with the contact details(if any) using left outer join.

• Select e.empno, e.fname, e.lname, e.dept, c.cntc\_type, c.cntc\_data from emp\_mstr e left join cntc\_dtls c on e.empno=c.codeno;

* Select e.empno, e.fname, e.lname, e.dept, c.cntc\_type, c.cntc\_data from emp\_mstr e cntc\_dtls c where e.empno=c.codeno(+);

All the employee details have to be listed even though their corresponding contact information is not present. This indicates all the rows from the first table will be displayed even though there exists no matching rows in the second table.

##### Right outer join

List the employee details with contact details(if any using right outer join.

* Tables Emp\_mstr(empno,fname,lname,dept) Cntc\_dtls(codeno,cntc\_type,cntc\_data)
* Select e.empno, e.fname, e.lname, e.dept, c.cntc\_type, c.cntc\_data from emp\_mstr e right join cntc\_dtls c on e.empno=c.codeno;
* Select e.empno, e.fname, e.lname, e.dept, c.cntc\_type, c.cntc\_data from emp\_mstr e cntc\_dtls c where e.empno(+)=c.codeno;

Since the RIGHT JOIN returns all the rows from the second table even if there are no matches in the first table.

##### Cross join

A cross join returns what known as a Cartesian Product. This means that the join combines every row from the left table with every row in the right table. As can be imagined, sometimes

join can be used in situation where it is desired, to select all possible combinations of rows & columns from both tables. The kind of join is usually not preferred as it may run for a very long time & produce a huge result set that may not be useful.

* Create a report using cross join that will display the maturity amounts for predefined deposits,

based on min & max period fixed/ time deposit.

* Tables Tem\_fd\_amt(fd\_amt) Fd\_mstr(minprd,maxprd,intrate)
* Select fd\_amt, s.minprd, s.maxprd, s.intrate,round (t.fd\_amt+(s.intrate/100 ) \* (s.minprd/365)

)) “amount\_min\_period”,round(t.fd\_amt+(s.intrate/100)\*(s.maxprd/365))) “amount\_max\_period” from fd\_mstr s cross join tem\_fd\_amt t;

* Select t.fd\_amt, s.minprd, s.maxprd, s.intrate, round(t.fd\_amt+(s.intrate/100) \* (s.minprd/365))) “amount\_min\_period”, round(t.fd\_amt+(s.intrate/100)\*(s.maxprd/365))) “amount\_max\_period”

from fd\_mstr s, tem\_fd\_amt t;

##### Self join

* + In some situation, it is necessary to join to itself, as though joining 2 separate tables.
  + This is referred to as self join Example
  + Emp\_mgr(empno,fname, lname,mgrno)
* Select e.empno,e.fname,e.lname, m.fname “manager” from emp\_mgr e, emp\_mgr m where e.mgrno=m.empno;

:

##### Employee ( Eno, Ename, Deptno, Salary ) Eno=pk, Deptno=fk Department ( Deptno, Dname ) Deptno=pk

Implement all join operation –cross join, natural join ,equi join, left outer ,right outer join etc & Write

SQL Queries for following questions

* 1. List of employee names of 'Computer' department.
  2. Find the Employee who‟sSalary above 50000 of each department.
  3. Find department name of employee name 'Amit'.

**Conclusion:** Thus we have studied to use & implement various join operation with nested queries.

### FAQ:

1. Explain Join Function.
2. Enlist the different types of join operations.
3. Explain CROSS Join explain with example.
4. Explain Natural join explain with example.
5. Explain Inner join explain with example.
6. Explain Outer join explain with example.
7. What is the use of nested Query. Explain with Example.

|  |  |
| --- | --- |
| **Assignment No.** | **4** |
| **Title** | Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory. |
| **PROBLEM STATEMENT/DEFINITION** | Write a PL/SQL code block to calculate the area of a circle for a value of radius varying from 5 to 9. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area. |
| **Objectives** | * Understand the control structure * Understand exception handling in PL/SQL |
| **Software packages and hardware apparatus used** | MySQL/Oracle  PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse |
| **References** | Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X  Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4  mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | * Date * Title * Problem Definition * Learning Objective * Learning Outcome * Theory-Related concept,Architecture,Syntax etc * Class Diagram/ER diagram * Test cases * Program Listing * Output * Conclusion |

### Assignment No: 4

**Title:-** Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory. Write a PL/SQL block of code for the following requirements:-

Schema:

1. Borrower(Rollin, Name, DateofIssue, NameofBook, Status)
2. Fine(Roll\_no,Date,Amt)

 Accept roll\_no & 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 & 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.

**Frame the problem statement for writing PL/SQL block inline with above statement. Objective:-** a)Understand the control structure b) Understand exception handling in PL/SQL

##### Theory:

**Introduction :-PL/SQL**

The development of database applications typically requires language constructs similar to those that can be found in programming languages such as C, C++, or Pascal. These constructs are necessary in order to implement complex data structures and algorithms. A major restriction of the database language SQL, however, is that many tasks cannot be accomplished by using only the provided language elements.

PL/SQL (Procedural Language/SQL) is a procedural extension of Oracle-SQL that o ffers language constructs similar to those in imperative programming languages.

Or

A PL/SQL is a procedural language extension to the SQL in which you can declare and use the variables, constants, do exception handling and you can also write the program modules in the form of PL/SQL subprograms.PL/SQL combines the features of a procedural language with structured query language

PL/SQL allows users and designers to develop complex database applications that require the usage of control structures and procedural elements such as procedures, functions, and modules.

The basic construct in PL/SQL is a block. Blocks allow designers to combine logically related (SQL-) statements into units. In a block, constants and variables can be declared, and variables can be used to store query results. Statements in a PL/SQL block include SQL statements, control structures (loops), condition statements (if-then-else), exception handling, and calls of other PL/SQL blocks.

PL/SQL blocks that specify procedures and functions can be grouped into packages. A package is similar to a module and has an interface and an implementation part. Oracle o ffers several predeﬁned packages, for example, input/output routines, ﬁle handling, job scheduling etc. (see directory

$ORACLE HOME/rdbms/admin).

Another important feature of PL/SQL is that it o ffers a mechanism to process query results in a tuple- oriented way, that is, one tuple at a time. For this, cursors are used. A cursor basically is a pointer to a query result and is used to read attribute values of selected tuples into variables. A cursor typically is

used in combination with a loop construct such that each tuple read by the cursor can be processed individually.

##### In summary, the major goals of PL/SQL are to

* Increase the expressiveness of SQL,
* Process query results in a tuple-oriented way,
* Optimize combined SQL statements,
* Develop modular database application programs,
* Reuse program code, and
* Reduce the cost for maintaining and changing applications

##### Advantages of PL/SQL:-

Following are some advantages of Pl/SQL

* 1. Support for SQL :-PL/SQL is the procedural language extension to SQL supports all the functionalities of SQL.
  2. Improved performance:- In SQL every statement individually goes to the ORACLE server, get processed and then execute. But in PL/SQL an entire block of statements can be sent to ORACLE server at one time, where SQL statements are processed one at atime.PL/SQL block statements drastically reduce communication between the application and ORACLE. This helps in improving the performance.
  3. Higher Productivity:- Users use procedural features to build applications.PL/SQL code is written in the form of PL/SQL block.PL/SQL blocks can also used in other ORACLE Forms, ORACLE reports. This code reusability increases the programmers productivity.
  4. Portability :- Applications written in PL/SQL are portable. We can port them from one environment to any computer hardware and operating system environment running ORACLE.
  5. Integration with ORACLE :-Both PL/SQL and ORACLE are SQL based.PL/SQL variables have datatypes native to the oracle RDBMS dictionary. This gives tight integration with ORACLE.

##### Features of PL/SQL:-

1. We can define and use variables and constants in PL/SQL.
2. PL/SQL provides control structures to control the flow of a program. The control structures supported by PL/SQL are if..Then, loop, for..loop and others.
3. We can do row by row processing of data in PL/SQL.PL/SQL supports row by row processing using the mechanism called cursor.
4. We can handle pre-defined and user-defined error situations. Errors are warnings and called as exceptions in PL/SQL.
5. We can write modular application by using sub programs.

##### The structure of PL/SQL program:-

The basic unit of code in any PL/SQL program is a block. All PL/SQL programs are composed of blocks. These blocks can be written sequentially.

##### The structure of PL/SQL block:-

**DECLARE**

##### Declaration section BEGIN

**Executable section EXCEPTION**

##### Exception handling section

**END;**

Where

1. Declaration section

PL/SQL variables, types, cursors, and local subprograms are defined here.

1. Executable section

Procedural and SQL statements are written here. This is the main section of the block. This section is required.

1. Exception handling section

Error handling code is written here

This section is optional whether it is defined within body or outside body of program.

##### Conditional statements and Loops used in PL/SQL

Conditional statements check the validity of a condition and accordingly execute a set of statements. The conditional statements supported by Pl/SQL is

##### IF..THEN

* 1. **IF..THEN..ELSE**

##### IF..THEN..ELSIF

1. IF..THEN

Syntax1:-

If condition THEN Statement list END IF;

1. IF..THEN..ELSE

Syntax 2:-

IF condition THEN

Statement list

ELSE

Statements

1. IF..THEN..ELSIF

END IF;

Syntax 3:-

If condition THEN

Statement list ELSIF condition THEN

Statement list

ELSE

Statement list

END IF; END IF;

1. **CASE Expression :**CASE expression can also be used to control the branching logic within PL/SQL blocks. The general syntax is

CASE

WHEN <expression> THEN <statements>; WHEN <expression> THEN <statements>;

.

. ELSE

<statements>; END CASE;

Here expression in WHEN clause is evaluated sequentially. When result of expression is TRUE, then corresponding set of statements are executed and program flow goes to END CASE.

**ITERATIVE Constructs :** Iterative constructs are used to execute a set of statements respectively. The iterative constructs supported by PL/SQL are follows:

##### SIMPLE LOOP

* 1. **WHILE LOOP**

##### FOR LOOP

1. The Simple LOOP : It is the simplest iterative construct and has syntax like: LOOP

Statements

END LOOP;

The LOOP does not facilitate a checking for a condition and so it is an endless loop. To end the iterations, the EXIT statement can be used.

LOOP

<statement list>

IF condition THEN EXIT;

END IF;

END LOOP;

The statements here is executable statements,which will be executed repeatedly until the condition given if IF..THEN evaluates TRUE.

1. THE WHILE LOOP

The WHILE…LOOP is a condition driven construct i.e the condition is a part of the loop construct and

not to be checked separately. The loop is executed as long as the condition evaluates to TRUE. The syntax is:-

WHILE condition LOOP Statements

END LOOP;

The condition is evaluated before each iteration of loop. If it evaluates to TRUE, sequence of statements are executed. If the condition is evaluated to FALSE or NULL, the loop is finished and the control resumes after the END LOOP statement.

1. THE FOR LOOP :The number of iterations for LOOP and WHILE LOOP is not known in advance. THE number of iterations depends on the loop condition. The FOR LOOP can be used to have a definite numbers of iterations.

The syntax is:-

Where

For loop counter IN [REVERSE] Low bound..High bound LOOP Statements;

End loop;

* loop counter –is the implicitly declared index variable as BINARY\_INTEGER.
* Low bound and high bound specify the number of iteration .
* Statements:-Are the contents of the loop

**EXCEPTIONS:-** Exceptions are errors or warnings in a PL/SQL program.PL/SQL implements error handling using exceptions and exception handler.

Exceptions are the run time error that a PL/SQL program may encounter. There are two types of exceptions

* 1. Predefined exceptions
  2. User defined exceptions

1. **Predefined exceptions:-** Predefined exceptions are the error condition that are defined by ORACLE. Predefined exceptions cannot be changed. Predefined exceptions correspond to common SQL errors. The predefined exceptions are raised automatically whenever a PL/SQL program violates an ORACLE rule.
2. **User defined Exceptions:-** A user defined exceptions is an error or a warning that is defined by the program.User defined exceptions can be define in the declaration section of PL/SQL block.

User defined exceptions are declared in the declarative section of a PL/SQL block. Exceptions have a type Exception and scope**.**

##### Syntax :

DECLARE

<Exception Name> EXCEPTION; BEGIN

….

RAISE <Exception Name>

…

EXCEPTION

WHEN <Exception name> THEN

<Action>

END;

##### Exception Handling

A PL/SQL block may contain statements that specify exception handling routines. Each error or warning during the execution of a PL/SQL block raises an exception. One can distinguish between two types of exceptions:

##### System deﬁned exceptions

* + **User deﬁned exceptions** (which must be declared by the user in the declaration part of a block where the exception is used/implemented)

System deﬁned exceptions are always automatically raised whenever corresponding errors or warnings occur. User deﬁned exceptions, in contrast, must be raised explicitly in a sequence of statements using raise <exception name>. After the keyword exception at the end of a block, user deﬁned exception

handling routines are implemented. An implementation has the pattern when <exception name> then <sequence of statements>;

The most common errors that can occur during the execution of PL/SQL programs are handled by system deﬁned exceptions. The table below lists some of these exceptions with their names and a short

description.

|  |  |  |
| --- | --- | --- |
| **Oracle Error** | **Equivalent Exception** | **Description** |
| ORA-0001 | DUP\_VAL\_ON\_INDEX | Unique constraint violated. |
| ORA-0051 | TIMEOUT\_ON\_RESOURSE | Time-out occurred while waiting for recourse |
| ORA-0061 | TRANSACTION\_BACKED\_OUT | The transaction was rolled back to  due to deadlock. |
| ORA-1001 | INVALID\_CURSOR | Illegal cursor operation. |
| ORA-1012 | NOT\_LOGGED\_ON | Not connected to Oracle. |
| ORA-1017 | LOGIN\_DENIED | Invalid username/passward |
| ORA-1403 | NO\_DATA\_FOUND | No data found. |
| ORA-1410 | SYS\_INVALID\_CURSOR | Conversion to a universal rowed failed. |
| ORA-1422 | TOO\_MANY\_ROWS | A SELECT…INTO statement  matches more than one row. |
| ORA-1476 | ZERO\_DIVIDE | Division by zero. |
| ORA-1722 | INVALID\_NUMBER | Conversion to a number failed. |
| ORA-6500 | STORAGE\_ERROR | Internal PL/SQL error raised if PL/SQL runs out of memory. |
| ORA-6501 | PROGRAM\_ERROR | Internal PL/SQL error. |
| ORA-6502 | VALUE\_ERROR | Truncation, arithmetic or conversion error. |
| ORA-6504 | ROWTYPE\_MISMATCH | Host cursor variable and PL/SQL cursor variable have incompatible row type |
| ORA-6511 | CURSOR\_ALREADY\_OPEN | Attempt to open a cursor that is  already open. |
| ORA-6530 | ACCESS\_INTO\_NULL | Attempt to assign values to the attributes of a NULL object. |
| ORA-6531 | COLLECTION\_IS\_NULL | Attempt to apply collection methods other than EXISTS to a NULL PL/SQL table or varray. |
| ORA-6532 | SUBSCRIPT\_OUTSIDE\_LIMIT | Reference to a nested table or varray  index outside the declared range. |
| ORA-6533 | SUBSCRIPT\_BEYOND\_COUNT | Reference to a nested table or varray index higher than the number of |

|  |  |  |
| --- | --- | --- |
|  |  | elements in the collection |
| ORA-6592 | CASE\_NOT\_FOUND | No matching WHEN clause in a CASE statement is found |
| ORA-30625 | SELF\_IS\_NULL | Attempt to call a method on a NULL object instance |

##### Syntax:-

**<Exception\_name>Exception;**

**Handling Exceptions:-** Exceptions handlers for all the exceptions are written in the exception handling section of a PL/SQL block.

Syntax:-

Exception

When exception\_name then Sequence\_of\_statements1;

When exception\_name then Sequence\_of\_statements2;

When exception\_name then Sequence\_of\_statements3;

End;

##### Example:

Declare

emp sal EMP.SAL%TYPE; emp no EMP.EMPNO%TYPE;

too\_high\_sal exception;

begin

select EMPNO, SAL into emp no, emp sal from EMP where ENAME = ‟KING‟;

if emp sal ∗1.05 > 4000 then raise too high sal

else update EMP set SQL . . . end if ;

exception

when NO DATA FOUND – – no tuple selected then rollback;

when too\_high\_sal then insert into high sal emps values(emp no); commit;

end;

After the keyword when a list of exception names connected with or can be speciﬁed. The last when clause in the exception part may contain the exception name others. This introduces the default exception handling routine, for example, a rollback.

If a PL/SQL program is executed from the SQL\*Plus shell, exception handling routines may contain statements that display error or warning messages on the screen. For this, the procedure raise application error can be used. This procedure has two parameters <error number> and <message text>.

<error number> is a negative integer deﬁned by the user and must range between -20000 and -20999.

<error message> is a string with a length up to 2048 characters.

The concatenation operator “||” can be used to concatenate single strings to one string. In order to display numeric variables, these variables must be converted to strings using the function to char. If the

procedure raise application error is called from a PL/SQL block, processing the PL/SQL block terminates and all database modiﬁcations are undone, that is, an implicit rollback is performed in addition to displaying the error message.

Example:

if emp sal ∗1.05 > 4000

then raise application error(-20010, ‟Salary increase for employee with Id ‟|| to char (EMP no) ||‟ is too high‟);

##### E.g.

Declare

V\_maxno number (2):=20;

V\_curno number (2); E\_too\_many\_emp exception;

Begin

Select count (empno)into v\_curno from emp Where deptno=10;

If v\_curno>25 then Raise e\_too\_many\_Emp; End if;

Exception

when e\_too\_many\_emp then

….

…..

end;

**Lab Exercise**

1. Write a PL/SQL block to calculate factorial. Use Exception Handling.
2. Write a PL/SQL block to find prime number for first 30 numbers.
3. Write a PL/SQL block to find Fibonacci series for first 50 numbers.
4. Write a PL/SQL block to find **a** raised to power **b i.e. a b**
5. Write a PL/SQL block to find the grade of a student. Enter marks for 5 subjects.
6. Write a PL/SQL block to update the table. **Table: ACCT\_MSTR ==>**

|  |  |
| --- | --- |
| ACCT\_NO | CURBAL |
| SB1 | 500 |
| SB5 | 500 |
| SB9 | 500 |
| SB13 | 500 |

1. Write on your own one PL/SQL block for the problem statement.

#### FAQ :

* 1. What is PL/SQL? Explain.
  2. What is the difference between "SQL" and "PL/SQL"?
  3. What are the different Goals of PL/SQL?
  4. What are exceptions? What are the different types of exceptions?
  5. What are the different conditional statements used in PL/SQL?
  6. What are the different iterative construct used in PL/SQL? Explain in short.
  7. What are the features of PL/SQL? Explain.
  8. What are the advantages of PL/SQL? Explain
  9. How will you stop an infinite loop without closing the program?
  10. Why PL/SQL does not support retrieving multiple records?

|  |  |
| --- | --- |
| **Assignment No.** | **5** |
| **Title** | Write a PL/SQL stored procedure and function. |
| **PROBLEM STATEMENT/DEFINITION** | Write a Stored Procedure namely proc\_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks899and 825 category is Higher Second Class. |
| **Objectives** | * Understand IF-THEN condition and FOR loop * Understand the PL/SQL Stored Procedure. * Understand the PL/SQL Stored Function * Write PL/SQL block code using stored procedure and stored function. |
| **Software packages and hardware apparatus used** | MySQL/Oracle  PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse |
| **References** | Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X  Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4  mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | * Date * Title * Problem Definition * Learning Objective * Learning Outcome * Theory-Related concept,Architecture,Syntax etc * Class Diagram/ER diagram * Test cases * Program Listing * Output * Conclusion |

### Assignment No: 5

**Title** :- PL/SQL Stored Procedure and Stored Function Write a Stored Procedure namely proc\_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class. Write a PL/SQL block for using procedure created with above requirement. Stud\_Marks(name, total\_marks) Result(Roll,Name, Class). **Frame the separate problem statement for writing PL/SQL Stored Procedure and function, inline with above statement. The problem statement should clearly state the requirements.**

**Objective:-** a) Understand IF-THEN condition and FOR loop b) Understand the PL/SQL Stored Procedure

c) Understand the PL/SQL Stored Function d) Write PL/SQL block code.

##### Theory :- PROCEDURE:-

A procedure is a subprogram that performs a specific action or task. A procedure has two parts.

1. The procedure specification: The procedure specification specifies the procedure name and the parameters it accepts. It is not necessary to create a procedure that accepts parameters.
2. The procedure body: The procedure body contains the declarative section without DECLARE keyword, the executable section and an exception section.

##### Syntax for creating a procedure

**Create [or replace] PROCEDURE procedure\_name [(argument1 [IN / OUT / IN OUT] type), (argument2 [IN / OUT / IN OUT] type),**

##### ….]

**IS/AS**

##### Procedure\_body

Where

Procedure\_name: – is the name of the procedure to be created Argument:- is the name of the procedure parameter

Type:- Is the data type of the associated parameter

Procedure\_body:-Is a PL/SQL block that makes up the code of the procedure.

IN:-This is default mode. The value of the actual parameter is passed into the procedure. Inside the procedure the formal parameter is considered read only.

OUT:-Any value the actual parameter has when the procedure is called ignored. Inside the procedure ,the formal parameters are considered as write only.

IN OUT:-this mode is combination of IN and OUT

**Deleting procedure:-** To remove a procedure from the database.

##### Syntax:-

**Drop procedure<procedure\_name>;**

##### FUNCTION:-

A function is a subprogram, which is used to compute values. It is similar to a procedure, function also takes arguments and can be in different modes. Function also can be stored in the database. It is a PL/SQL block consisting of declarative, executable and exception section.

Difference between procedure and function is that the procedure call is a PL/SQL statement by itself, while a function call is called as a part of an expression.

A function can return more than one value using OUT parameter. A function can be called using positional or named notation.

##### Syntax for creating a function:-

**Create [or replace] FUNCTION function\_name [(argument1 [IN / OUT / IN OUT] type), (argument2 [IN / OUT / IN OUT] type),**

##### ….]

**Return return\_type IS / AS**

Where

##### Function\_body

Function\_name: – is the name of the function to be created Argument: - is the name of the function parameter

Type:- Is the data type of the associated parameter

Function\_body:-Is a PL/SQL block containing code for the function.

IN:-This is default mode. The value of the actual parameter is passed into the procedure. Inside the procedure the formal parameter is considered read only.

OUT:-Any value the actual parameter has when the procedure is called ignored. Inside the procedure ,the formal parameters are considered as write only.

INOUT:-this mode is combination of IN and OUT

**Deleting a Function:-** To remove the subprogram from the database.

##### Syntax:-

**Drop function<function\_name>;**

**Package :**

A package is a PL/SQL construct that allows related objects to be stored together. A package has 2 separate parts: the specification and the body. Each of them stored separately in the data dictionary. **Package Specification** :

CREATE OR REPLACE PACKAGE package\_name

{IS|AS}

type\_definition|

procedure\_specification |Function specification| variable\_declaration |

exception\_declaration | cursor\_declaration | pragma declaration | end [procedure\_name];

##### Package Body:

The package body is separate data dictionary object from the package header. It cannot be successfully compiled unless the package header has already been successfully compiled.

Syntax:

CREATE OR REPLACE PACKAGE BODY package\_name AS Procedure definition;

Function definition;

…….

End package\_name

To drop the package(both specification & the body) use the **drop package** command as follows: Syntax :

Drop package <package\_name>;

#### Lab Exercise

* 1. Write a procedure on EMP table. It should increase commission of an employee. Employee number and commission are passed as parameters to the called procedure.
  2. Write a function that returns the number of employees working in a department. Pass department number as an input to the function.
  3. Create table classes with the following fields

(Deptno, course, cur\_student, max\_student) Insert 4 or 5 records and

Write a function which returns true if the specified class is 80 percent full or more, and false otherwise. Write a PL/SQL block to call this function and use cursor in PL/SQL block to hold the records of all department.

* 1. Write a procedure to update records of classes table and write a PL/SQL block to call that procedure.
  2. Create a package which consist of procedures for insert ,delete and update the data of classes table.

#### FAQ :

1. Explain the term procedure and function of PL/SQL in short.
2. What is the difference between "procedure" and "function"?
3. What is the difference between "%type" and "%rowtype"?
4. What is package? Explain.
5. What is the use of package?
6. What are the different modes of argument passing?
7. What is difference between IN & IN OUT?
8. Write a package which consists of cursor, trigger, procedure & function.
9. What are the advantages of procedure & function?
10. Write the syntax to drop function, procedure & package.

|  |  |
| --- | --- |
| **Assignment No.** | **6** |
| **Title** | Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor) |
| **PROBLEM STATEMENT/DEFINITION** | Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table |
| **Objectives** | * Understand the types of cursors * Understand how to use cursors with PL/SQL block |
| **Software packages and hardware apparatus used** | MySQL/Oracle  PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse |
| **References** | Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X  Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4  mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | * Date * Title * Problem Definition * Learning Objective * Learning Outcome * Theory-Related concept,Architecture,Syntax etc * Class Diagram/ER diagram * Test cases * Program Listing * Output * Conclusion |

### Assignment No: 6

**Title** :- 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 Cust\_New with the data available in the table Cust\_Old. If the data in the first table already exist in the second table then that data should be skipped. Frame the separate problem statement for writing PL/SQL block to implement all types

**Objective :-** a)Understand the types of cursors b) Understand how to use cursors with PL/SQL block

##### Theory :-

##### CURSOR:-For the processing of any SQL statement, database needs to allocate memory. This memory is called context area. The context area is a part of PGA (Process global area) and is allocated on the oracle

##### server. A cursor is associated with this work area used by ORACLE, for multi row queries. A cursor is a handle or pointer to the context area .The cursor allows to process contents in the context area row by row.

There are two types of cursors.

1. Implicit cursor:-Implicit cursors are defined by ORACLE implicitly. ORACLE defines implicit cursor for every DML statements.
2. Explicit cursor:-These are user-defined cursors which are defined in the declaration section of the PL/SQL block. There are four steps in which the explicit cursor is processed.
   1. Declaring a cursor
   2. Opening a cursor
   3. Fetching rows from an opened cursor
   4. Closing cursor

##### General syntax for CURSOR:- DECLARE

**Cursor cursor\_name IS select\_statement or query; BEGIN**

##### Open cursor\_name;

**Fetch cursor\_name into list\_of\_variables; Close cursor\_name;**

##### END;

Where

1. Cursor\_name:-is the name of the cursor.
2. Select\_statement:-is the query that defines the set of rows to be processed by the cursor.
3. Open cursor\_name:-open the cursor that has been previously declared. When cursor is opened following things happen
   1. The active set pointer is set to the first row.
   2. The value of the binding variables are examined.
4. Fetch statement is used to retrieve a row from the selected rows, one at a time, into PL/SQL variables.
5. Close cursor\_name:-When all of cursor rows have been retrieved, the cursor should be closed.

##### Explicit cursor attributes:-

Following are the cursor attributes

1. **%FOUND**: - This is Boolean attribute. It returns TRUE if the previous fetch returns a row and false if it doesn‟t.
2. **%NOTFOUND**:-If fetch returns a row it returns FALSE and TRUE if it doesn‟t. This as the exit condition for the fetch loop;

is often used

1. **%ISOPEN**:-This attribute is used to determine whether or not the associated cursor is open. If so it returns TRUE otherwise FALSE.
2. **%ROWCOUNT**:-This numeric attribute returns a number of rows fetched by the cursor.

##### Cursor Fetch Loops

1. **Simple Loop Syntax:-**

LOOP

Fetch cursorname into list of variables;

EXIT WHEN cursorname%NOTFOUND Sequence\_of\_statements;

END LOOP;

##### WHILE Loop Syntax:-

1. **Cursor FOR Loop Syntax:**

FETCH cursorname INTO list of variables; WHILE cursorname%FOUND LOOP

Sequence\_of\_statements;

FETCH cursorname INTO list of variables; END LOOP;

FOR variable\_name IN cursorname LOOP

-- an implicit fetch is done here.

-- cursorname%NOTFOUND is also implicitly checked.

-- process the fetch records. Sequence\_of\_statements;

END LOOP;

There are two important things to note about :-

* 1. Variable\_name is not declared in the DECLARE section. This variable is implicitly declared by the PL/SQL compiler.
  2. Type of this variable is cursorname%ROWTYPE.

##### Implicit Cursors

PL/SQL issues an implicit cursor whenever you execute a SQL statement directly in your code, as long as that code does not employ an explicit cursor. It is called an "implicit" cursor because you, the developer, do not explicitly declare a cursor for the SQL statement.

If you use an implicit cursor, Oracle performs the open, fetches, and close for you automatically; these actions are outside of your programmatic control. You can, however, obtain information about the most recently executed SQL statement by examining the values in the implicit SQL cursor attributes.

PL/SQL employs an implicit cursor for each UPDATE, DELETE, or INSERT statement you execute in a program. You cannot, in other words, execute these statements within an explicit

cursor, even if you want to. You have a choice between using an implicit or explicit cursor only when you execute a single-row SELECT statement (a SELECT that returns only one row).

In the following UPDATE statement, which gives everyone in the company a 10% raise, PL/SQL creates an implicit cursor to identify the set of rows in the table which would be affected by the update:

UPDATE employee

SET salary = salary \* 1.1;

The following single-row query calculates and returns the total salary for a department. Once again, PL/SQL creates an implicit cursor for this statement:

SELECT SUM (salary) INTO department\_total FROM employee

WHERE department\_number = 10;

If you have a SELECT statement that returns more than one row, you must use an explicit cursor for that query and then process the rows returned one at a time. PL/SQL does not yet support any kind of array interface between a database table and a composite PL/SQL datatype such as a PL/SQL table.

##### Drawbacks of Implicit Cursors

Even if your query returns only a single row, you might still decide to use an explicit cursor. The implicit cursor has the following drawbacks:

* It is less efficient than an explicit cursor
* It is more vulnerable to data errors
* It gives you less programmatic control

The following sections explore each of these limitations to the implicit cursor.

**Inefficiencies of implicit cursors**

An explicit cursor is, at least theoretically, more efficient than an implicit cursor. An implicit cursor executes as a SQL statement and Oracle's SQL is ANSI-standard. ANSI dictates that a single-row query must not only fetch the first record, but must also perform a second fetch to determine if too many rows will be returned by that query (such a situation will RAISE the TOO\_MANY\_ROWS PL/SQL exception). Thus, an implicit query always performs a minimum of two fetches, while an explicit cursor only needs to perform a single fetch.

This additional fetch is usually not noticeable, and you shouldn't be neurotic about using an implicit cursor for a single-row query (it takes less coding, so the temptation is always there). Look out for indiscriminate use of the implicit cursor in the parts of your application where that cursor will be executed repeatedly. A good example is the Post-Query trigger in the Oracle Forms.

Post-Query fires once for each record retrieved by the query (created from the base table block and the criteria entered by the user). If a query retrieves ten rows, then an additional ten fetches are needed with an implicit query. If you have 25 users on your system all performing a similar query, your server must process 250 additional (unnecessary) fetches against the database. So, while it might be easier to write an implicit query, there are some places in your code where you will want to make that extra effort and go with the explicit cursor.

**Vulnerability to data errors**

If an implicit SELECT statement returns more than one row, it raises the TOO\_MANY\_ROWS exception. When this happens, execution in the current block terminates and control is passed to the exception section. Unless you deliberately plan to handle this scenario, use of the implicit cursor is a declaration of faith. You are saying, "I trust that query to always return a single row!"

It may well be that today, with the current data, the query will only return a single row. If the nature of the data ever changes, however, you may find that the SELECT statement which formerly identified a single row now returns several. Your program will raise an exception. Perhaps this is what you will want. On the other hand, perhaps the presence of additional records is inconsequential and should be ignored.

With the implicit query, you cannot easily handle these different possibilities. With an explicit query, your program will be protected against changes in data and will continue to fetch rows without raising exceptions.

##### Lab Exercise

1. Create table with name **student** having the field **rollno**, **first name**, **last name** & **branch**. Insert 10 records into table. Write a PL/SQL to create a cursor to hold all the record of student table having branch „**Computer Science**‟. Display all the records.
2. Write a PL/SQL block to update the record of rollno =100 & set the **branch** to **E and TC’**, if it is not present then insert the record into the student table with the id=100; (use implicit cursor sql%notfound).
3. Write a cursor and use it to raise the employee salaries as follows:
   1. All employees of department 20 get 5% raise
   2. All employees of department 30 get 10% raise
   3. Rest of employees get 7.5% raise Use separate cursor.

**FAQ :**

1. What is cursor?
2. What are the different types of cursors?
3. What are the different attributes of explicit cursor? Explain in brief.
4. What is implicit cursor?
5. Explain the FOR loop of Cursor.
6. What is difference between simple loop, while loop & for loop?
7. What is difference between Implicit & Explicit Cursor?
8. Explain FOR UPDATE cursor with an example.
9. What is CURRENT OF clause in cursor? Give an example.
10. List all predefined cursor.

|  |  |
| --- | --- |
| **Assignment No.** | **7** |
| **Title** | Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers). |
| **PROBLEM STATEMENT/DEFINITION** | Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library\_Audit table |
| **Objectives** | * Understand the concept of row level and statement level trigger * Understand the concept of trigger initiated against event. |
| **Software packages and hardware apparatus used** | MySQL/Oracle  PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse |
| **References** | Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X  Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4  mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | * Date * Title * Problem Definition * Learning Objective * Learning Outcome * Theory-Related concept,Architecture,Syntax etc * Class Diagram/ER diagram * Test cases * Program Listing * Output * Conclusion |

### Assignment No: 7

**Title** :- Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers). Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library\_Audit table.

**Objective :-** a)Understand the concept of row level and statement level trigger

b) Understand the concept of trigger initiated against event

##### Theory :- DATABASE TRIGGERS:-

A database trigger is a PL/SQL program unit, which gets fired automatically whenever the data event such as DML or DDL system event. Triggers are associated with a specific table and are fired automatically whenever the table gets manipulated in a predefined way. The act of executing a trigger is called as firing a trigger.

Triggers are similar to procedures in that they are named PL/SQL blocks with declarative, executable and exception handling sections. But the difference is a procedure is executed explicitly from another block via a procedure call but a trigger is executed implicitly whenever the triggering event happens. A

procedure can pass arguments but trigger doesn‟t accept arguments

A database trigger has following components:-

1.A triggering **Event**

2.A triggering **Constraint**

3.A triggering **Action Trigger categories**

Triggers are categorized in various ways.

1)Trigger type 2)Triggering time 3)Triggering event

##### Trigger types

There are two types of triggers

1. **Statement Trigger**:-A statement trigger is a trigger in which the trigger action is executed once for the manipulation operation that fires the trigger.
2. **Row Trigger**:-A row trigger is a trigger in which the trigger action is performed repeatedly for each row of the table that is affected by the manipulation operation that fires the trigger.

##### Triggering time

Triggers can specify the time of trigger action.

##### Before the triggering event

The trigger action is performed before the operation that fires the trigger is executed. This trigger is used when execution of operation depends on trigger action.

##### After the triggering event

The trigger action is performed after the operation that fires the trigger is executed. This trigger is used when triggering action depends on the execution of operation. **Triggering Events**

Triggering events are the DML operations. These operations are **insert, update and delete** When these operations are performed on a table, the trigger which is associated with the operation is fired.

Triggering events divide triggers into three types.

* 1. DELETE TRIGGER
  2. UPDATE TRIGGER
  3. INSERT TRIGGER

##### General syntax for creation of Trigger

**Create [or replace] TRIGGER <trigger\_name>**

##### <BEFORE | AFTER>

**DELETE | [OR] INSERT | [OR] UPDATE[OF <column1>[,<column2>…..]**

##### ON <table\_name>

**[for each row[when <condition>] Begin**

##### ……… ………

**………………. End;**

Where

Trigger\_name:-trigger name is the name of the trigger. Table\_name :-is thye table name for which the trigger is defined.

Trigger-condition:-The trigger condition in the when clause,if present is evaluated first.The body of the trigger is executed only when this condition evaluates to true.

##### Dropping trigger

Suppose you want to drop trigger then the syntax is

Syntax:-Drop trigger trigger\_name;

##### Enabling and Disabling Triggers

The Trigger can be disabled without dropping them. When the trigger is disabled, it is still exists in data dictionary but never fired, To disable trigger, use alter command.

Syntax:-

Alter TRIGGER trigger\_name DISABLE/ENABLE; For all triggers on a particular table

Syntax:-

Alter TRIGGER trigger\_name (DISABLE/ENABLE) all triggers;

**Lab Exercise :-**

1. Create a trigger that audits the operations on an Emp table. Steps

Create table emp\_audit

(id number, operation varchar2(6), Dt date, User\_id number, Username varchar2(20));

If any operation like insert, update, delete done on EMP table then insert into EMP\_audit table information like the name of the operation with id, user\_id and date.

1. Create a table Employee(id, Emp\_name, Salary, City)

Create a trigger to convert the Emp\_name into upper case before inserting or updating on Employee table.

1. Create a trigger to check Salary is less than 20000 before inserting or updating on Employee table.
2. Create a trigger (Statement Level Trigger) to display messages after inserting or updating or deleting records on Employee Table.

#### FAQ :

1. Write a database Trigger
2. Explain Database Trigger Components.
3. Explain Trigger Types with e.g.
4. Explain difference between Row-Level & Statement-Level Trigger.
5. Write a Syntax for Enable & Disable Trigger.
6. Write a Syntax for Displaying Trigger Errors.

|  |  |
| --- | --- |
| **Assignment No.** | **8** |
| **Title** | **Database Connectivity:** |
| **PROBLEM STATEMENT/DEFINITION** | Write a program to implement MySQL/Oracle database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.) |
| **Objectives** | Insert a record in mysql database using Java/PHP.  update a record in mysql database using Java/PHP. |
| **Software packages and hardware apparatus used** | MySQL/Oracle  PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse |
| **References** | Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X  Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4  http://docs.mongodb.org/manual |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | * Date * Title * Problem Definition * Learning Objective * Learning Outcome * Theory-Related concept,Architecture,Syntax etc * Class Diagram/ER diagram * Test cases * Program Listing * Output * Conclusion |

|  |  |
| --- | --- |
| **Assignment No.** | **9** |
| **Title** | **MongoDB Queries:**  Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators) |
| **PROBLEM STATEMENT/DEFINITION** | DesignandDevelopMongoDBQueriesusingCRUDoperations.(UseCRUDoperations, SAVE method, logical operators etc.). |
| **Objectives** | * Understand the concept of Binary JSON format. * Understand the concept of Mongo DB document model. |
| **Software packages and hardware apparatus used** | MongoDB  PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse |
| **References** | 1.Kristina Chodorow,Michael Dierolf,”MongoDB:The definite  Guide”, O’Reilly Publications,ISBN:978-1-449-34468-9.  2.Kevin Roebuck,”Storing and Managing Big Data-  NoSQL,Hadoop and More”,Emereopty  Limited,ISBN:1743045743,9781743045749  http://docs.mongodb.org/manual |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | * Date * Title * Problem Definition * Learning Objective * Learning Outcome * Theory-Related concept,Architecture,Syntax etc * Class Diagram/ER diagram * Test cases * Program Listing * Output * Conclusion |

# Assignment No. 09

**Aim :** Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators)

* **Objectives :** Understand the concept of Binary JSON format.

Understand the concept of Mongo DB document model.

**Theory** : **MongoDB** is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

##### Database

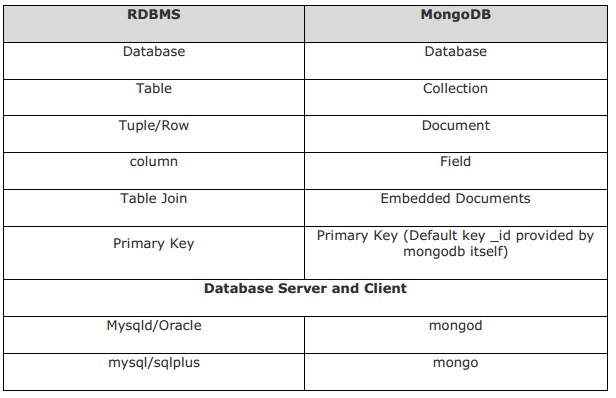
Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

##### Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

##### Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB .

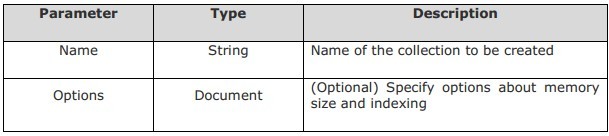
CRUD is the basic operation of Mongodb ,it stands CREATE , READ , UPDATE, DELETE.

**MongoDB ─ 1. Create Collection**

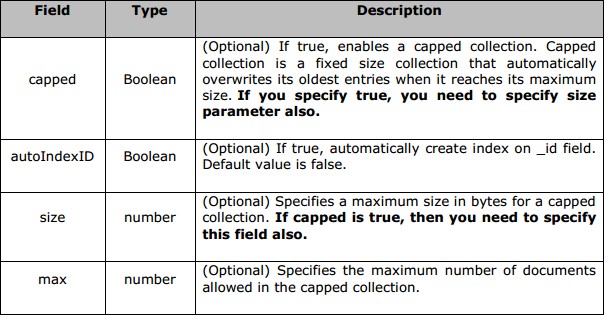
The createCollection() Method

MongoDB db.createCollection(name, options) is used to create collection. Basic syntax of createCollection() command is as follows: **db.createCollection(name, options)**

In the command, name is name of collection to be created. Options are a document and are used to specify configuration of collection.



Options parameter is optional, so you need to specify only the name of the collection. Following is the list of options you can use:



While inserting the document, MongoDB first checks size field of capped collection, then it checks max field.

Examples

Basic syntax of createCollection() method without options is as follows:

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

>

You can check the created collection by using the command show collections.

>show collections mycollection

system.indexes

#### READ-The find() Method

To query data from MongoDB collection, you need to use MongoDB's find()method. Syntax

The basic syntax of find() method is as follows:

>db.COLLECTION\_NAME.find()

find()method will display all the documents in a non-structured way. The pretty() Method

To display the results in a formatted way, you can use pretty() method. Syntax

>db.mycol.find().pretty() Example

>db.mycol.find().pretty()

{

"\_id": ObjectId(7df78ad8902c), "title": "MongoDB Overview",

"description": "MongoDB is no sql database", "by": "tutorials point",

"url": "[http://www.tutorialspoint.com](http://www.tutorialspoint.com/)", "tags": ["mongodb", "database", "NoSQL"], "likes": "100"

}

>

Apart from find() method, there is findOne() method, that returns only one document.

#### UPDATE

MongoDB's update() and save() methods are used to update document into a collection.

The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

MongoDB Update() Method

The update() method updates the values in the existing document. The basic syntax of update() method is as follows:

##### >db.COLLECTION\_NAME.update(SELECTIOIN\_CRITERIA, UPDATED\_DATA)

**Example**

Consider the mycol collection has the following data.

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"} Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}})

##### >db.mycol.find()

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

>

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

>db.mycol.update({'title':'MongoDB Overview'},

{$set:{'title':'New MongoDB Tutorial'}},{multi:true})

#### MongoDB Save() Method

The save() method replaces the existing document with the new document passed in the save() method.

The basic syntax of MongoDB save() method is −

##### >db.COLLECTION\_NAME.save({\_id:ObjectId(),NEW\_DATA}) Example

Following example will replace the document with the \_id '5983548781331adf45ec7'.

>db.mycol.save(

{

"\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point New Topic",

"by":"Tutorials Point"

} )

>db.mycol.find()

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"Tutorials Point New Topic", "by":"Tutorials Point"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

#### DELETE-The remove() Method

MongoDB's remove() method is used to remove a document from the collection.

remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

 deletion criteria: (Optional) deletion criteria according to documents will be removed.

 justOne: (Optional) if set to true or 1, then remove only one document. Basic syntax of remove() method is as follows:

##### >db.COLLECTION\_NAME.remove(DELLETION\_CRITTERIA)

**Example**

Consider the mycol collection has the following data.

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"} Following example will remove all the documents whose title is 'MongoDB Overview'.

>db.mycol.remove({'title':'MongoDB Overview'})

>db.mycol.find()

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

#### LOGICAL OPERATORS:

**Syntax**

#### AND in MongoDB

In the find() method, if you pass multiple keys by separating them by ',' then MongoDB treats it as AND condition. Following is the basic syntax of AND −

>db.mycol.find({key1:value1, key2:value2}).pretty() Example

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

>db.mycol.find({"by":"tutorials point","title": "MongoDB Overview"}).pretty()

{

"\_id": ObjectId(7df78ad8902c), "title": "MongoDB Overview",

"description": "MongoDB is no sql database", "by": "tutorials point",

"url": "[http://www.tutorialspoint.com](http://www.tutorialspoint.com/)", "tags": ["mongodb", "database", "NoSQL"], "likes": "100"

}>

For the above given example, equivalent where clause will be ' where by='tutorials point' AND title = 'MongoDB Overview' '. You can pass any number of key, value pairs in find clause.

#### OR in MongoDB

**Syntax :** To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR −

>db.mycol.find( { $or: [ {key1: value1}, {key2:value2} ] } ).pretty()

Example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{ "\_id": ObjectId(7df78ad8902c), "title": "MongoDB Overview",

"description": "MongoDB is no sql database", "by": "tutorials point",

"url": "[http://www.tutorialspoint.com](http://www.tutorialspoint.com/)", "tags": ["mongodb", "database", "NoSQL"], "likes": "100" }

##### Using AND and OR Together Example

The following example will show the documents that have likes greater than 100 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is 'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'

##### >db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": ObjectId(7df78ad8902c), "title": "MongoDB Overview",

"description": "MongoDB is no sql database", "by": "tutorials point",

"url": "[http://www.tutorialspoint.com](http://www.tutorialspoint.com/)", "tags": ["mongodb", "database", "NoSQL"], "likes": "100" }

**Conclusion:** Thus we have studied MongoDB Queries using CRUD operations.

#### FAQ:-

1. Explain CREATE Operation with example.
2. Explain AND Operator with example.
3. Explain DELETE function in Mongodb.
4. Explain DELETE function in Mongodb.
5. Explain FIND function in Mongodb.
6. Explain OR Operator with example.

|  |  |
| --- | --- |
| **Assignment No.** | **10** |
| **Title** | **MongoDB Aggregation and Indexing:**  Implement aggregation and indexing with suitable example  using MongoDB. |
| **PROBLEM STATEMENT/DEFINITION** | Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB |
| **Objectives** | Understand indexing and aggregation concept on MongoDB |
| **Software packages and hardware apparatus used** | Mongodb  Operating Systems   * (64-Bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards * Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, MongoDB 2.6. |
| **References** | 1. MongoDB: The Definitive Guide, 2nd Edition,Powerful and Scalable Data Storage By Kristina Chodorow  Publisher: O'Reilly Media 2. http://docs.mongodb.org/manual |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | * Date * Title * Problem Definition * Learning Objective * Learning Outcome * Theory-Related concept,Architecture,Syntax etc * Class Diagram/ER diagram * Test cases * Program Listing * Output * Conclusion |

# Assignment No. 10

**Aim :** Implement aggregation and indexing with suitable example using MongoDB.

**Objectives :** Understand indexing and aggregation concept on MongoDB records

**Theory :** MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++. This tutorial will give you great understanding on MongoDB concepts needed to create and deploy a highly scalable and performance-oriented database.

**Aggregations** operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count(\*) and with group by is an equivalent of mongodb aggregation.

The aggregate() Method For the aggregation in MongoDB, you should use aggregate() method. Basic syntax of aggregate() method is as follows:

>db.COLLECTION\_NAME.**aggregate**(AGGREGATE\_OPERATION)

**Example**

In the collection you have the following data:

{

\_id: ObjectId(7df78ad8902c) title: 'MongoDB Overview',

description: 'MongoDB is no sql database', by\_user: 'tutorials point',

url: 'http://www.tutorialspoint.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100

},

{

\_id: ObjectId(7df78ad8902d)

title: 'NoSQL Overview',

description: 'No sql database is very fast', by\_user: 'tutorials point',

url: 'http://www.tutorialspoint.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 10

},

{

\_id: ObjectId(7df78ad8902e) title: 'Neo4j Overview',

description: 'Neo4j is no sql database', by\_user: 'Neo4j',

url: 'http://www.neo4j.com',

tags: ['neo4j', 'database', 'NoSQL'], likes: 750

},

Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following aggregate() method:

**> db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum :1}}}])**

{

"result" : [

{

"\_id" : "tutorials point", "num\_tutorial" : 2

},

{

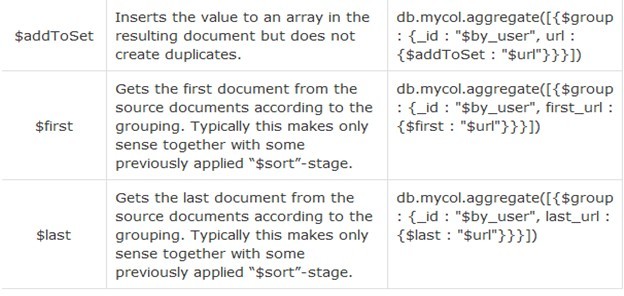
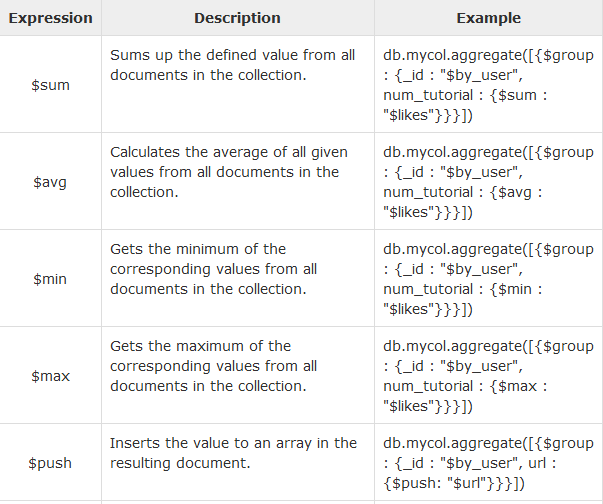
"\_id" : "Neo4j","num\_tutorial" : 1

}],

"ok" : 1

}**>**

Sql equivalent query for the above use case will be select by\_user, count(\*) from mycol group by by\_user.



Pipeline Concept

In UNIX command, shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also supports same concept in

aggregation framework. There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn be used for the next stage and so on.

Following are the possible stages in aggregation framework:

 $project: Used to select some specific fields from a collection.

 $match: This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.

$group: This does the actual aggregation as discussed above.

 $sort: Sorts the documents.

 $skip: With this, it is possible to skip forward in the list of documents for a given amount of documents.

 $limit: This limits the amount of docu ments to look at, by the given number starting from the current positions.

 $unwind: This is used to unwind document that are using arrays. When using an array, the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

**Indexes** support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement This scan is highly inefficient and require MongoDB to process a large volume of data.

Indexes are special data structures, that store a small portion of the data set in an easy -to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

##### The ensureIndex() Method

To create an index you need to use ensureIndex() method of MongoDB. The basic syntax of ensureIndex() method is as follows().

>db.COLLECTION\_NAME.ensureIndex({KEY:1})

Here key is the name of the file on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

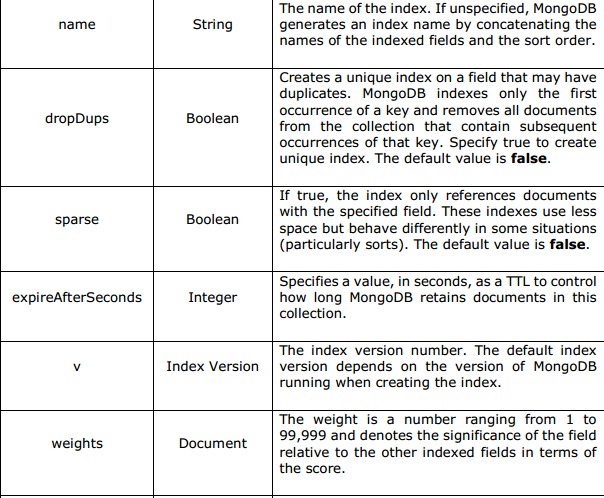
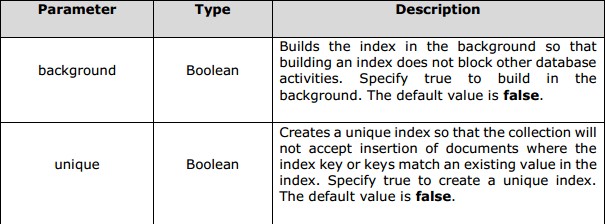
##### Example

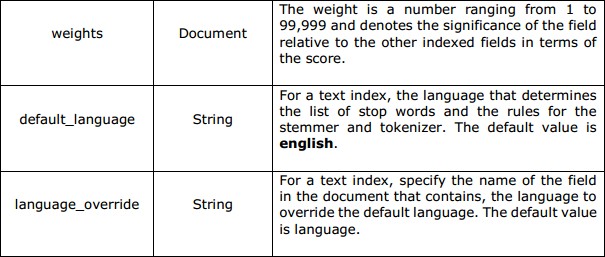
>db.mycol.ensureIndex({"title":1})

In **ensureIndex()** method you can pass multiple fields, to create index on multiple fields.

>db.mycol.ensureIndex({"title":1,"description":-1})

ensureIndex() method also accepts list of options (which are optional). Following is the list:





**Conclusion: -** *Thus we have studied use and implementation of aggregation function &indexing function.*

**FAQ : -**

1. Enlist various aggregation operations.
2. Explain MIN function with example.
3. Explain PUSH function with example.
4. Explain SUM & AVG function with example.

|  |  |
| --- | --- |
| **Assignment No.** | **11** |
| **Title** | **MongoDB Map-reduces operations:**  Implement Map reduces operation with suitable example  using MongoDB |
| **PROBLEM STATEMENT/DEFINITION** | Implement Map reduces operation with suitable example using MongoDB |
| **Objectives** | * To understand concept of Map-reduce as data processing paradigm for condensing large volumes of data into useful aggregated results. |
| **Software packages and hardware apparatus used** | Operating Systems  (64-Bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards  Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, TC++, GTK++, mongoDB 2.6. |
| **References** | <http://docs.mongodb.org/manual>   * MongoDB: The Definitive Guide, 2nd Edition,Powerful and Scalable Data Storage By Kristina Chodorow  Publisher: O'Reilly Media |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | 1. Title 2. Problem Definition 3. Learning Objectives 4. Theory 5. Class Diagram/ER Diagram 6. Test cases 7. Program Listing 8. Output 9. Conclusion |

# Assignment No. 11

**Aim :** Implement Map reduces operation with suitable example using MongoDB

**Objectives :** To understand concept of Map-reduce as data processing paradigm for condensing large volumes of

data into useful aggregated results

**Theory :**As per the MongoDB documentation, MapReduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses mapReduce command for map-reduce operations. MapReduce is generally used for processing large data sets.

##### MapReduce Command

Following is the syntax of the basic mapReduce command

>db.collection.mapReduce (

function() { emit(key,value); }, //map function function(key,values) {return reduceFunction},

{ //reduce function

out: collection, query: document, sort: document, limit: number

} )

The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values. In the above syntax –

 map is a javascript function that maps a value with a key and emits a key-value pair

 reduce is a javascript function that reduces or groups all the documents having the same key

 out specifies the location of the map -reduce query result

 query specifies the optional selection criteria for selecting d ocuments

 sort specifies the optional sort criteria

 limit specifies the optional maximum number of documents to be returned Using MapReduce Consider the following document structure storing user posts. The document stores user\_name of the

user and the status of post.

{"post\_text": "tutorialspoint is an awesome website for tutorials" , "user\_name": "mark",

"status":"active" }

We will use a mapReduce function on our posts collection to select all the active posts, group them on the basis of user\_name and then count the number of posts by each user using the following code

##### >db.posts.mapReduce(

**function() { emit(this.user\_id,1); },**

##### function(key, values) {return Array.sum(values)}, { query:{status:"active"},

**out:"post\_total" })**

The above mapReduce query outputs the following result −

{

"result" : "post\_total", "timeMillis" : 9,"counts" :

{

"input" : 4,

"emit" : 4,

"reduce" : 2,

"output" : 2

},

"ok" : 1,

}

The result shows that a total of 4 documents matched the query (status:"active"), the map function emitted 4 documents with key-value pairs and finally the reduce function

grouped mapped documents having the same keys into 2.

To see the result of this mapReduce query, use the find operator −

##### >db.posts.mapReduce ( function() { emit(this.user\_id,1); }, function(key, values) {return Array.sum(values)}, {query:{status:"active"}, out:"post\_total"}).find()

The above query gives the following result which indicates that both users tom and mark have two posts in active states −

{ "\_id" : "tom", "value" : 2 }

{ "\_id" : "mark", "value" : 2 }

In a similar manner, MapReduce queries can be used to construct large complex aggregation queries. The use of custom Javascript functions make use of MapReduce which is very flexible and powerful.

**Conclusion:** *Thus we have studied Map reduce function.*

**FAQ : -**

1. Define and Explain mapreduce in MongoDB with examples.
2. Why to use Mapreduce in MongoDB
3. Explain the structure of ObjectID in MongoDB.
4. What are NoSQL databases? What are the different types of NoSQL databases?

|  |  |
| --- | --- |
| **Assignment No.** | **12** |
| **Title** | **Database Connectivity:** |
| **PROBLEM STATEMENT/DEFINITION** | Write a program to implement Mongo DB database connectivity with any front end language to implement Database navigation operations(add, delete, edit etc.) |
| **Objectives** | * Understand the concept of Connectivity between Java and databases * Understand how Java can invoke CRUD operation. |
| **Software packages and hardware apparatus used** | Operating Systems  (64-Bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards  Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, TC++, GTK++, mongoDB 2.6. |
| **References** | <http://docs.mongodb.org/manual>   * MongoDB: The Definitive Guide, 2nd Edition,Powerful and Scalable Data Storage By Kristina Chodorow  Publisher: O'Reilly Media |
| **STEPS** | Refer to details |
| **Instructions for writing journal** | 1. Title 2. Problem Definition 3. Learning Objectives 4. Theory 5. Class Diagram/ER Diagram 6. Test cases 7. Program Listing 8. Output 9. Conclusion |

# Assignment No. 12

|  |  |
| --- | --- |
| **Mini Project.** | **13 Group C Mini Project :** |
| **Title** | Using the **database concepts covered in Group A and Group B**, develop an application with following details:   1. Follow the same problem statement decided in Assignment -1 of Group A. 2. Follow the Software Development Life cycle and other concepts learnt in **Software Engineering Course** throughout the implementation. 3. Develop application considering:  * Front End: Java/Perl/PHP/Python/Ruby/.net/any other language * Backend : MongoDB/ MySQL/Oracle |
| **PROBLEM STATEMENT/DEFINITION** |  |
| **Objectives** |  |
| **Software packages and hardware apparatus used** |  |
| **References** |  |
| **STEPS** |  |
| **Instructions for writing journal** |  |

**Subject Coordinator Head Computer Engg. Department**

**(Mrs.Pranjali P. Joshi) (Dr.Geetanjali V. Kale)**