

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
**M.V.S.R. ENGINEERING COLLEGE**  
(Sponsored by Matrusri Education Society, Estd. 1980)  
Affiliated to Osmania University & Recognized by AICTE  
Saroornagar (M), Nadargul(PO), Hyderabad - 501510

**DATABASE MANAGEMENT SYSTEMS LAB**  
( Course Code : PC 263 CS )



# Certificate

Department of COMPUTER SCIENCE & ENGINEERING

Certified that this is a bonafide work of lab experiments carried out by

Mr/Ms. \_\_\_\_\_ bearing

Roll.No. \_\_\_\_\_ under

the course **DATABASE MANAGEMENT SYSTEMS** Laboratory (PC 263 CS)

prescribed by Osmania University for **B.E. Sem-IV(AICTE)** of

**COMPUTER SCIENCE & ENGINEERING**

during the academic year \_\_\_\_\_ – \_\_\_\_\_.

Internal Examiner

External Examiner

## INDEX

[illegible]

## PC 263 CS: DATABASE MANAGEMENT SYSTEMS LAB

Cumulative Record  
(Lab Manual & Observation Book)

Compiled By:-  
K.Murali Krishna, N.Sabitha & G.Srishailam  
Assistant Professor(s), CSED, MVSREC

### SQL (Structured Query Language):

Structured Query Language is a database computer language designed for managing data in relational database management systems(RDBMS), and originally based upon Relational Algebra. Its scope includes data query and update, schema creation and modification, and data access control. SQL was one of the first languages for Edgar F.Codd's relational model in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks" and became the most widely used language for relational databases. IBM developed SQL in the mid 1970's. Oracle incorporated in 1979. SQL used by IBM/DB2 and DS Database Systems. SQL was adopted as the standard language for RDBS by ANSI in 1989.

SQL language is subdivided into several language elements, including:

- Clauses, which are in some cases optional, constituent components of statements and queries.
- Expressions, which can produce either scalar values or tables consisting of columns and rows of data.

- Predicates which specify conditions that can be evaluated to SQL three-valued logic (3VL) Boolean truth values and which are used to limit the effects of statements and queries, or to change program flow.
- Queries which retrieve data based on specific criteria.
- Statements which may have a persistent effect on schemas and data, or which may control transactions, program flow, connections, sessions, or diagnostics.
- SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.
- Insignificant white space is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

There are five types of SQL statements. They are:

1. DATA DEFINITION LANGUAGE (DDL)
2. DATA MANIPULATION LANGUAGE (DML)
3. DATA RETRIEVAL LANGUAGE (DRL)
4. TRANSACTIONAL CONTROL LANGUAGE (TCL)
5. DATA CONTROL LANGUAGE (DCL)

### DATA DEFINITION LANGUAGE (DDL):

The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases

of a database project. Let's take a look at the structure and usage of four basic DDL commands:

1. CREATE 2. ALTER 3. DROP 4. RENAME

### **DATA MANIPULATION LANGUAGE (DML):**

The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

1. INSERT 2. UPDATE 3. DELETE

### **DRL(DATA RETRIEVAL LANGUAGE):**

Retrieves data from one or more tables.

1. SELECT

### **TRANSACTIONAL CONTROL LANGUAGE (TCL):**

A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be made permanent to the database only if they are committed. A transaction begins with an executable SQL statement & ends explicitly with either rollback or commit statement.

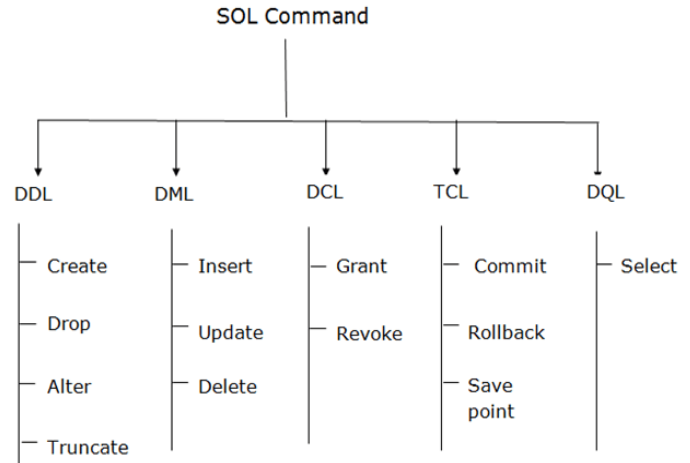
1. COMMIT 2. ROLLBACK

### **DATA CONTROL LANGUAGE (D.C.L): DCL**

provides users with privilege commands. The owner of database objects (tables), has the sole authority over them. The owner (database administrators) can allow other database users to access the objects as per

their requirement

1. GRANT 2. REVOKE



## **Program 1**

### **CREATION OF DATABASE**

#### **Problem Definition**

Creation of database (exercising the commands for creation)

#### **Problem Description**

Creating the structure of an entity is done by using DDL commands. Data Definition Language (DDL) commands are those commands that define either the structure of an object or the database in which the objects are stored. These commands include the database creation, modification and administration.

**CREATE TABLE:** This command defines all characteristics of a table, including integrity and check constraints, column names and data types, storage option, and so on.

Data in relational models is stored in tables. So the first step is creating a table using SQL command **CREATE TABLE**. Before a table is created the following factors are to be finalized.

- The name of the table
- The number of columns in the table
- The name, data type and size of each column of the table
- The column and table constraints.

#### **Pseudo code**

##### **Select a database**

To select a particular database to work with you issue the **USE** statement as follows:

```
USE <database_name>;
```

##### **Create a new database**

```
CREATE DATABASE [IF NOT EXISTS]  
<database_name>;
```

##### **Review the created database**

```
SHOW DATABASE <database_name>;
```

##### **Delete an existing database in the server.**

```
DROP DATABASE [IF EXISTS]  
<database_name>;
```

#### **Display Databases**

Show databases;

#### **Create Table Syntax**

```
CREATE TABLE table_name ({column  
datatype [column_constraints] |  
table_constraints } );
```

```
CREATE TABLE table_name(  
column1datatype,  
column2datatype,  
columnNdatatype,  
  
PRIMARY KEY( one or more columns ),  
Foreign Key( column ) References  
tablename(column)  
);
```

**ALTER    TABLE** <table\_name>    **ADD**  
<column\_name datatype>;

**ALTER    TABLE** <table\_name>    **MODIFY**  
**COLUMN** <column\_name datatype>;

**ALTER    TABLE** <table\_name>    **DROP**  
**COLUMN** <column\_name>;

### **To list tables**

Show tables;

### **TRUNCATE:**

The SQL TRUNCATE TABLE command is used to delete complete data from an existing table. The basic syntax of a TRUNCATE TABLE command is as follows.

*TRUNCATE TABLE table\_name;*

### **TASK-1: Creation, altering and dropping tables using SQL(DDL) Commands**



## Program 2

### SIMPLE AND COMPLEX QUERIES

#### Problem Definition

Simple to complex condition query creation using SQL Plus

#### Problem Description

SQL Plus commands are used fundamentally for accessing and managing the data in the database.

**Simple Queries:** Simple Queries includes insert, select, update, alter, delete, drop, savepoint, commit, rollback etc.

**Complex Queries:** Complex Queries includes Joins, Subqueries, constraints, views etc.

#### Pseudo Code

##### Read data from one or more tables

```
SELECT column1, column2, columnN FROM  
table_name;
```

```
SELECT * FROM < table_name>;
```

##### To insert new records in a table

```
INSERT INTO <TABLE_NAME> VALUES  
(value1,value2,value3,...valueN);
```

```
INSERT INTO <TABLE_NAME> (column1, col  
umn2, ...) VALUES (value1, value2, ...);
```

##### To delete data from a table

```
DELETE FROM <table_name> WHERE  
[condition];
```

```
UPDATE <table_name> SET column1 = value1,  
column2 = value, columnN = valueN WHERE  
[condition];
```

#### Transactional Control Language

```
SAVEPOINT <savepoint-name>;
```

```
ROLLBACK;
```

```
COMMIT;
```

#### Integrity Constraints

```
ALTER TABLE <table_name> ADD  
CONSTRAINT <MyPrimaryKey> PRIMARY  
KEY (column1, column2.);
```

```
ALTER TABLE <table_name> ADD  
CONSTRAINT <MyCheckConstraint> CHECK  
(CONDITION);
```

```
ALTER TABLE <table_name> ADD  
CONSTRAINT <MyUniqueConstraint> UNIQUE  
(column1,column2....);
```

```
ALTER TABLE <table_name> DROP  
CONSTRAINT <constraint-name>;
```

```
SELECT <column_name> [, column_name ]  
FROM <table1> [, table2 ] WHERE  
<column_name> OPERATOR (SELECT  
<column_name> [, column_name ] FROM table1  
[, table2 ] [WHERE]);
```

```
SELECT <column1, column2> FROM <table1,  
table2> WHERE [conditions] GROUP BY
```



column1, column2 HAVING [conditions]  
ORDER BY column1, column2;

**Order by :** The order by clause is used to display the results in sorted order.

**Group by :** The attribute or attributes given in the clauses are used to form groups. Tuples with the same value on all attributes in the group by clause are placed in one group.

**Having:** SQL applies predicates (conditions) in the having clause after groups have been formed, so aggregate function be used.

Set Operations: UNION - OR INTERSECT - AND EXCEPT - - NOT

NESTED QUERY:- A nested query makes use of another sub-query to compute or retrieve the information.

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Various Aggregate Functions

1. Count()
2. Sum()
3. Avg()
4. Min()
5. Max()

An operator is a reserved word or a character used primarily in an SQL statement WHERE

clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

1. Arithmetic operators
2. Comparison operators
3. Logical operators
4. Operators used to negate conditions

## JOINS

SELECT <column\_name(s)> FROM <table1>  
INNER JOIN <table2> ON  
table1.column\_name=table2.column\_name;

SELECT <column\_name(s)> FROM <table1>  
LEFT JOIN/RIGHT JOIN <table2> ON  
table1.column\_name=table2.column\_name;  
SELECT <column\_name(s)> FROM <table1>  
FULL OUTER JOIN <table2> ON  
table1.column\_name=table2.column\_name;

**CREATE VIEW** <view\_name> AS SELECT  
<column\_name(s)> FROM <table\_name>  
WHERE [condition];

### **Task-2.1: Write MySQL queries which use**

#### Select

- distinct
- order by
- group by

#### Comparison operators

- Like, Not like
- >, <, >=, <=, ==, <>
- Between and, not between and
- In, not in
- Any, all

#### Aggregate functions

- Count, Sum, Average, Max, Min

#### Set Operations

- Union, Except, Intersect

#### Nested Queries

- Correlated





**Task-2.2:working of various kinds of join operations in MySQL such as**

- Cartesian Product
- Inner Join
  - Self Join
  - Equi join / Natural join
  - Non equi join
- Outer Join
  - Left
  - Right

### Program 3

## TRIGGERS

#### Problem Definition

Using Triggers and stored procedures.

#### Problem Description

Database Triggers are procedures that are stored in the database and are explicitly executed (fired) where the contents of a table are changed.

Triggers have 3 basic parts.

1. Trigger event: It is a SQL statement that causes the trigger to be fired. It can insert, delete or update statements for a specified table.
2. Trigger restriction: It specifies a Boolean (logical) expression that must be true for the trigger to fire. It is specified using a where clause.
3. Trigger action: It is a procedure that contains the SQL statement and PL / SQL code to be executed when a triggering statement is issued and the trigger restriction evaluation to true.

#### Types of triggers

**Row trigger:** A row trigger is fired each time the table is affected by a triggering statement. Eg: if an update statement updates multiple rows of a table, a row trigger is fired once for each row affected by the update statement.

**Statement trigger:** A row trigger is fired once on behalf of the triggering statement independent of

the number of rows the triggering statement affects.

Eg: If the trigger makes the security check on the time or the user.

**Before trigger:** It executes the trigger action before the triggering statement is executed. Eg: Before triggers are used to derive specific column values before completing a triggering INSERT or UPDATE statement.

**After trigger:** It executes the trigger after the triggering statement is executed. After triggers are used when you want the triggering statement to complete before executing the triggering action.

#### Pseudo code

Create or replace trigger [Schema.]

<Triggername>

{ Before | After }

{ Delete | Insert | Update [of column, -----] } On  
[Schema.] <tablename>

[Referencing { OLD as old, NEW as new } ] for  
each row [ when condition ]

Raise\_application\_error (error-no, 'message');

#### Syntax of deleting a trigger

Drop trigger trigger-name;

#### CURSOR:

We have seen how oracle executes an SQL statement. Oracle DBA uses a work area for its internal processing. This work area is private to SQL's operation and is called a cursor.

The data that is stored in the cursor is called the Active Data set. The size of the cursor in memory is the size required to hold the number of rows in the Active Data Set.

**Explicit Cursor-** You can explicitly declare a cursor to process the rows individually.

A cursor declared by the user is called Explicit Cursor. For Queries that return more than one row, You must declare a cursor explicitly.

The data that is stored in the cursor is called the Active Data set. The size of the cursor in memory is the size required to hold the number of rows in the Active

**Why use an Explicit Cursor-** Cursor can be used when the user wants to process data one row at a time.

**Explicit Cursor Management-** The steps involved in declaring a cursor and manipulating data in the active data set are:-

- Declare a cursor that specifies the SQL select statement that you want to process.
- Open the Cursor.
- Fetch the data from the cursor one row at a time.
- Close the cursor.

**Explicit Cursor Attributes-** Oracle provides certain attributes/ cursor variables to control the execution of the cursor. Whenever any cursor(explicit or implicit) is opened and used, Oracle creates a set of four system variables via

which Oracle keeps track of the 'Current' status of the cursor. You

- Declare a cursor that specifies the SQL select statement that you want to process.
- Open the Cursor.
- Fetch the data from the cursor one row at a time.
- Close the cursor.

#### **How to Declare the Cursor:-**

The General Syntax to create any particular cursor is as follows:-

*Cursor <Cursorname> is Sql Statement;*

#### **How to Open the Cursor:-**

The General Syntax to Open any particular cursor is as follows:-

*Open Cursorname;*

#### **Fetching a record From the Cursor:-**

The fetch statement retrieves the rows from the active set to the variables one at a time.

Each time a fetch is executed. The focus of the DBA cursor advances to the next row in the Active set.

One can make use of any loop structure(Loop-End Loop along with While,For) to fetch the records from the cursor into one row at a time.

The General Syntax to Fetch the records from the cursor is as follows:-

*Fetch cursorname into variable1,variable2,\_\_\_\_\_*

#### **Closing a Cursor:-**

The General Syntax to Close the cursor is as follows:-

*Close <cursorname>;*

**Task 3.1:**

**Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:**

ID	NAME	AGE	ADDRESS	SALARY
1	Alive	24	Khammam	2000
2	Bob	27	Kadappa	3000
3	Catri	25	Guntur	4000
4	Dena	28	Hyderabad	5000
5	Eeshwar	27	Kurnool	6000
6	Farooq	28	Nellur	7000

**Task:3.2**

**Write a PL/SQL code that updates the balance of the account holder of 'Delhi' branch and displays the number of records updated. (Use implicit cursor)**

**Table :** Accounts(AcCNo, Name, Balance, DateofOpening, Branch)





## **Program -4**

### **SUB PROGRAMS**

A procedure or function that has been stored in the library cache is referred to as a stored procedure or a stored function. A stored procedure or stored function has the following characteristics:

- It has a name: This is the name by which the stored procedure or function is called and referenced.
- It takes parameters: These are the values sent to the stored procedure or function from the application.
- It returns values: A stored procedure or function can return one or more values based on the purpose of the procedure or function.

### **PROCEDURES**

A procedure is a one kind of subprogram, which is designed and created to perform a specific operation on data in your database. A procedure takes zero or more input parameters and returns zero or more output parameters.

The syntax of a creation of procedure is as follows:

#### **Syntax:**

```
CREATE OR REPLACE PROCEDURE
procedure_name [(argument1 [IN/OUT/IN OUT]
datatype,
argument2 [IN/OUT/IN OUT] datatype,...)] IS
[<local variable declarations>]
BEGIN
[EXCEPTION
```

Optional Exception Handler(s) ]

Executable Statements

END;

The procedure is made up of two parts: the declaration and the body of the procedure. The declaration begins with the keyword **PROCEDURE** and ends with the last parameter declaration. The body begins with the keyword **IS** and ends with the keyword **END**.

The procedure body is further divided into three parts : declarative, executable and exception parts are the same as PL/SQL blocks. Exceptions/Errors are handled using **DECLARE EXIT/CONTINUE FOR <<error>> stmt;**

The declaration section is used to assign names and define parameter list, which variables are passed to the procedure and which values are returned from the procedure back to the calling program.

Parameters can be defined in the following format **Argument [parameter mode] datatype**. There are three types of parameters mode: **IN**, **OUT** and **IN OUT**

#### **IN Mode**

- Default parameter mode.
- Used to pass values to the procedure.
- Formal parameter can be a constant, literal, initialized variable or expression
- Used for reading purpose

**OUT Mode**

- Used to return values to the caller.
- Formal parameters cannot be used in an expression, but should be assigned a value.
- Used for writing purpose

**IN OUT Mode**

- Used to pass values to the procedure as well as return values to the caller
- Formal parameter acts like an initialized variable and should be assigned a value.
- Used for both reading and writing purpose

**The EXCEPTION Section**

In both procedures and functions, you can add optional exception handlers. These exception handlers allow you to return additional information based on certain conditions (such as no data found or some user-specified condition). By using exception handlers and allowing the stored procedure to notify you of some special conditions, you can minimize the amount of return-value checking that must be done in the application code. Because the work to determine that no data has been selected has already been done by the RDBMS engine, you can save on resources if you take advantage of this information. Exceptions/Errors are handled using DECLARE EXIT/CONTINUE FOR <<error>> stmt;

**Functions:**

A function, like a procedure, is a set of PL/SQL statements that form a subprogram. The

subprogram is designed and created to perform a specific operation on data. A function takes zero or more input parameters and returns just one output value. If more than one output value is required, a procedure should be used. The syntax of a function is as follows:

**Syntax**

```
CREATE OR REPLACE Function function_name
[(argument1 [IN/OUT/IN OUT] datatype,
argument2 [IN/OUT/IN OUT] datatype,...)]
RETURN datatype IS
[<local variable declarations>]
BEGIN
[EXCEPTION
Optional Exception Handler(s)]
```

**PL/SQL Statements**

```
END [function_name];
/
```

As with a procedure, a function is made up of two parts: the declaration and the body. The declaration begins with the keyword Function and ends with RETURN statement. The body begins with the keyword IS and ends with the keyword END.

The difference between a procedure and a function is the return value. A function has the return declaration as well as a RETURN function within the body of that function that returns a value. This RETURN function is used to pass a return value to the calling program.

*Note: If you do not intend to return a value to the*

*calling program, or you want to return more than one value, use a procedure.*

**Task 4.1:** calculate the net salary and year salary if da is 30% of basic, hra is 10% of basic and pf is 7% if basic salary is less than 8000, pf is 10% if basic sal between 8000 to 160000

**Task 4.2:** Print Fibonacci series using functions.

## **Program 5**

### **FORM DESIGNING**

#### **Problem Definition**

Creation of Forms using LibreOffice Base.

LibreOffice Base is a database management program, similar to Microsoft Access.

LibreOffice Base is a front end database manager that can be used to create forms and reports from a variety of databases including MySQL as well as others.

LibreOffice Base can be used to create small embedded databases when used with Java-based HSQLDB as its storage engine (The LibreOffice Base default database). This makes LibreOffice Base appear as if it were a database manager and the database, but it is not. LibreOffice Base is just the front end allowing us to tie into the actual database.

#### **Steps for creating a Form**

The LibreOffice suite of tools includes a very powerful database application — one that happens to be incredibly user-friendly. These databases can be managed/edited by any user and data can be entered by anyone using a LibreOffice-generated form. These forms are very simple to create and can be attached to existing databases or you can create both a database and a form in one fell swoop.

There are two ways to create LibreOffice Base forms:

- ❖ Form Wizard
- ❖ Design View.

Design view is a versatile drag and drop form creator that is quite powerful and allows you to add elements and assign those elements to database tables. The Form Wizard is a very simple step-by-step wizard that walks the user through the process of creating a form. Although the Wizard isn't nearly as powerful as the Design View — it will get the job done quickly and doesn't require any form of design experience.

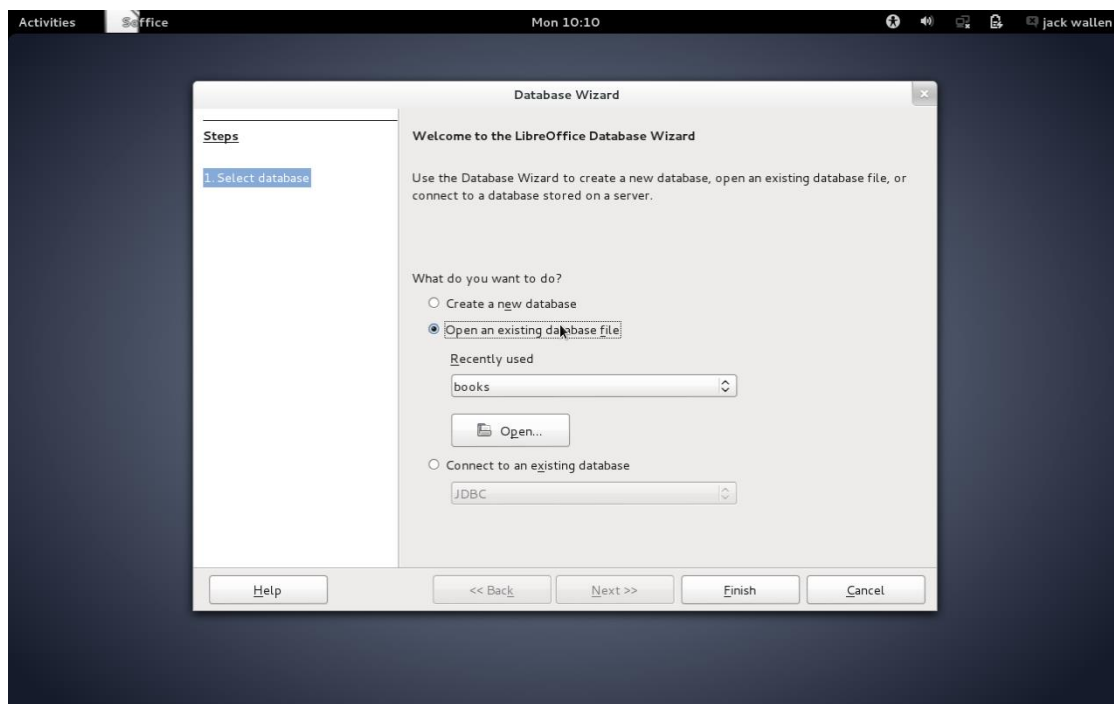
For this entry, I will address the Form Wizard (in a later post, I will walk you through the more challenging Design View). I will assume you already have a database created and ready for data entry. This database can either be created with LibreOffice and reside on the local system or be a remote database of the format:

1. Oracle JDBC
2. Spreadsheet
3. dBASE
4. Text
5. MySQL
6. ODBC.

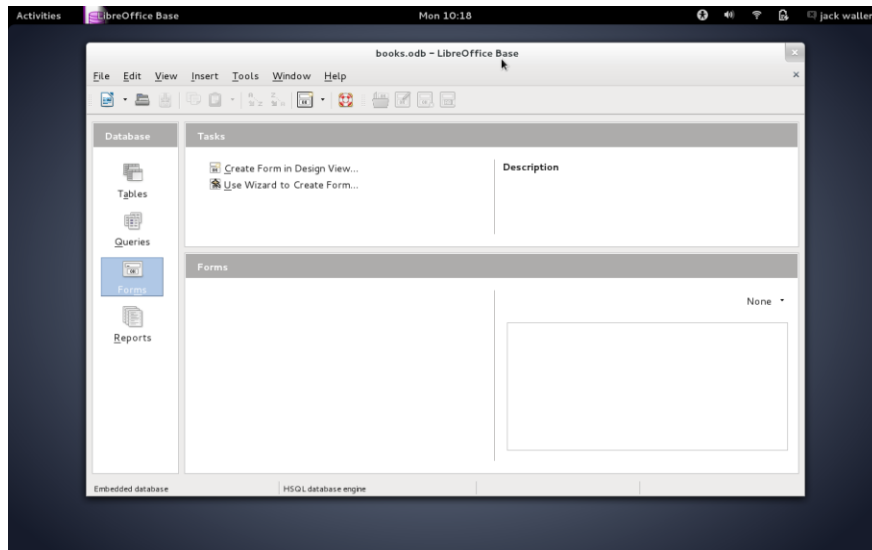
For purposes of simplicity, we'll go with a local LibreOffice Base-generated database. I've created a very simple database with two tables to be used for this process. Let's create a data entry form for this database.

### Opening the database

The first step is to open LibreOffice Base. When the Database Wizard window appears (Figure 1), select Open an existing database file, click the Open button, navigate to the database to be used, and click Finish

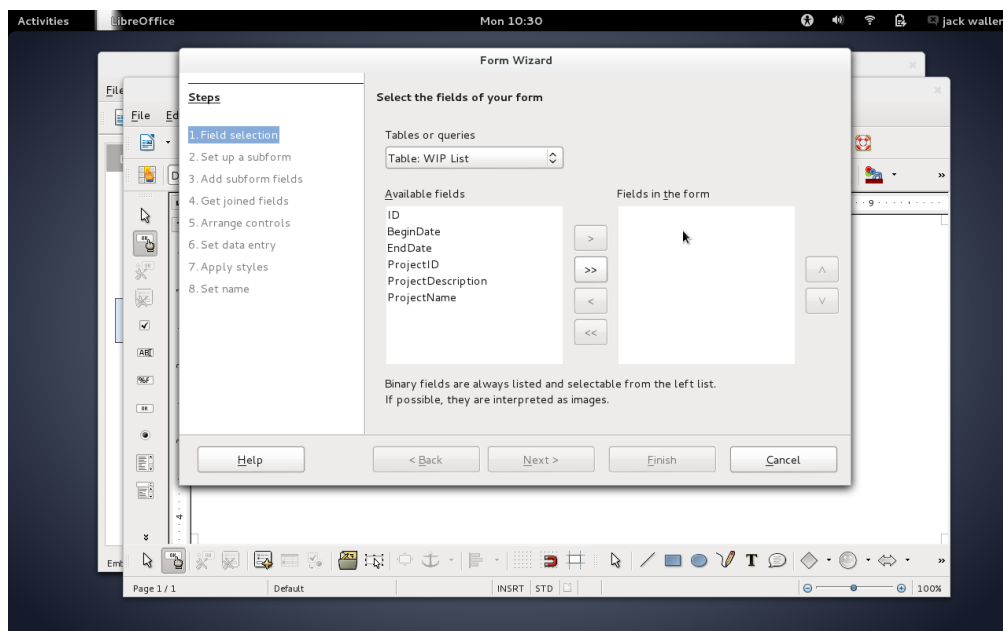


The next window to appear is the heart and soul of LibreOffice Base. Here in Figure you can manage tables, run queries, create/edit forms, and view reports of the opened database.



Click the Forms button in the left-side navigation and then double-click Use Wizard to Create Form under Tasks. When the database opens in the Form Wizard, your first step is to select the fields available to the form. You do not have to select all fields from the database. You can select them all or you can select as few as one.

If your database has more than one table, you can select between the tables in the Tables or queries drop-down (NOTE: You can only select fields from one table in the database at this point). Select the table to be used and then add the fields from the Available fields section to the Fields in the form section in the Figure 3.

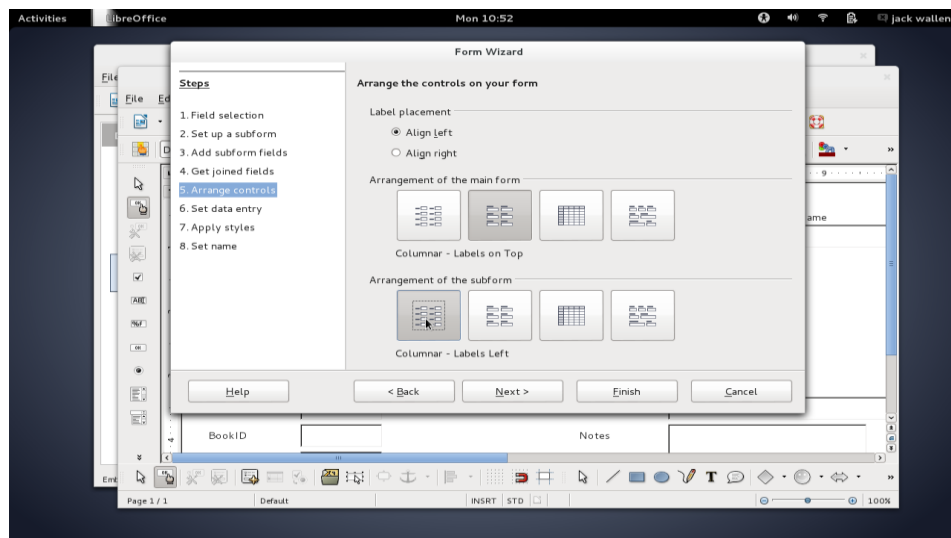


Once you've selected all the necessary fields, click Next. At this point, you can choose to add a subform. A subform is a form-within-a-form and allows you to add more specific data to the original form. For example, you can include secondary data for employee records (such as work history, raises, etc.) to a form. This is the point at which you can include fields from other tables (besides the initial table selected from the Tables or queries drop-down). If you opt to create a subform for your data, the steps include:

- Selecting the table
- Adding the fields
- Joining the fields (such as AuthorID to ID — Figure).

### Arrange form controls

After all subforms are added, click Next to continue on. In the next step, you must arrange the controls of the form. This is just another way of saying how you want the form to look and feel (where do you want the data entry field to reside against the field label). You can have different layouts for forms and subforms .



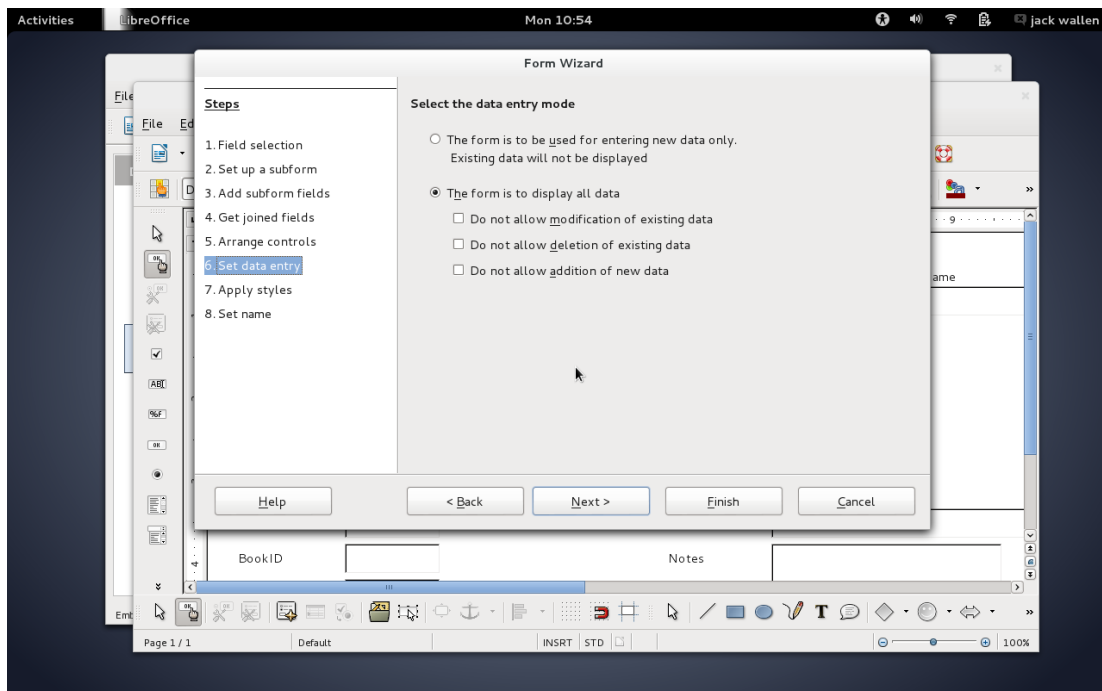
### Select data entry mode

Click Next when you've arranged your controls. The next step is to select the data entry mode (Figure 6). There are two data entry modes:



- Enter new data only
- Display all data.

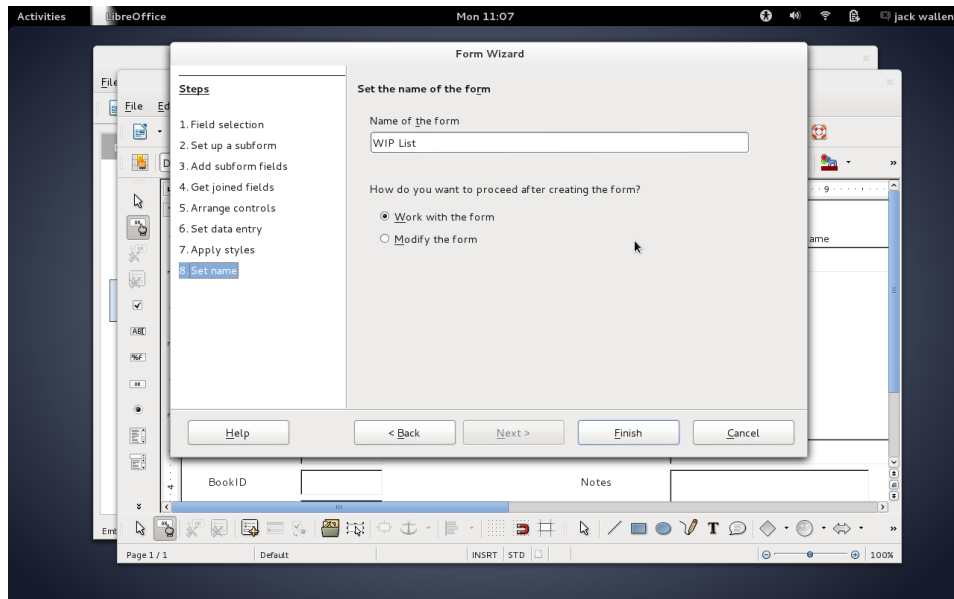
If you want to use the form only as a means to enter new data, select Enter new data only. If, however, you know you'll want to use the form to enter and view data, select Display all data. If you go for the latter option, you will want to select whether previously entered data can be modified or not. If you want to prevent write access to the previous data, select Do not allow modification of existing data.



Make your selection and click Next. Start entering data

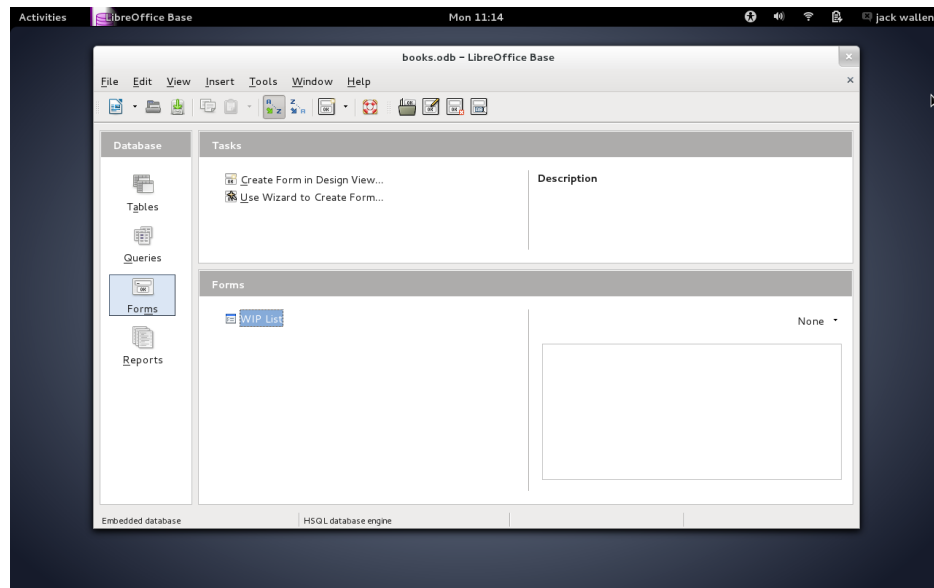
At this point you can select a style for your form. This allows you to pick a color and field border (no border, 3D border, or flat). Make your selection and click Next.

The last step is to name your form. In this same window you can select the option, immediately begin working with the form (Figure 7). Select that option and click Finish. At this point, your form will open and you can start entering data.



After a form is created, and you've worked with and closed said form ... how do you re-open a form to add more data? Simple:

1. Open LibreOffice Base.
2. Open the existing database (in the same manner you did when creating the form).
3. Double-click the form name under Forms (Figure 8).
4. Start entering data.



As a final note, make sure, after you finish working with your forms, that you click File > Save in the LibreOffice Base main window, to ensure you save all of your work.

You can create as many forms as you need with a single database — there is no limit to what you can do.

If you're looking to easily enter data into LibreOffice databases, creating user-friendly forms is just a few steps away. Next time we visit this topic, we'll walk through the Design View method of form creation.

## **Reports**

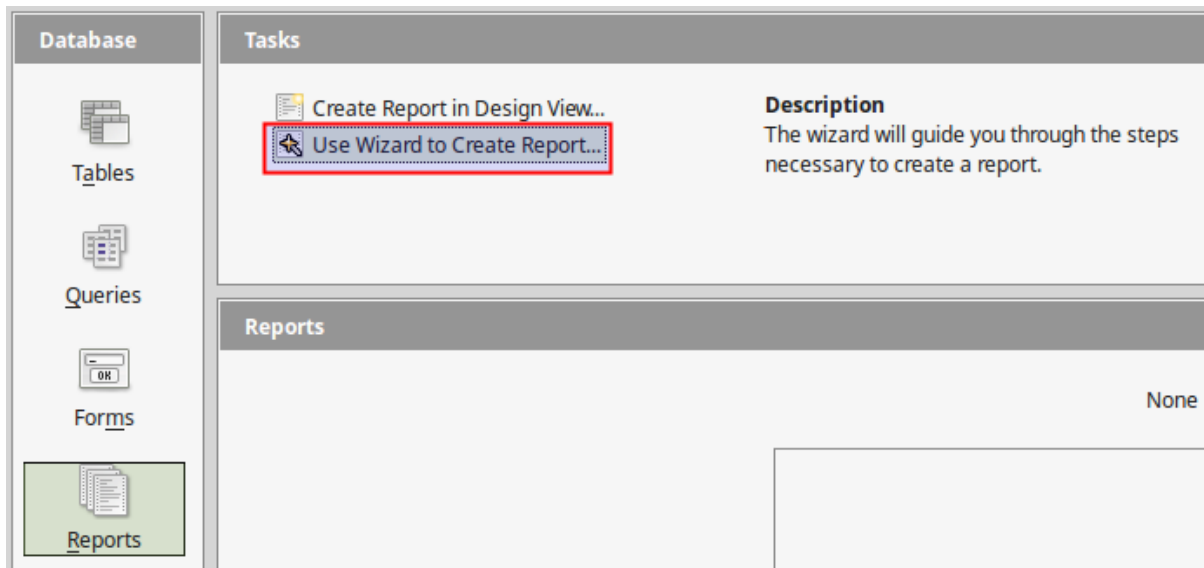
Reports give you the ability to present information from your database in an easy-to-read, visually appealing printable format. For example, you can create a simple report of phone numbers for all your contacts, or a summary report on the total sales across different regions and time periods. Base makes it easy to create and customize a report using data from any query or table in your database.

### Creating Reports

In LibreOffice Base you can create a Report using

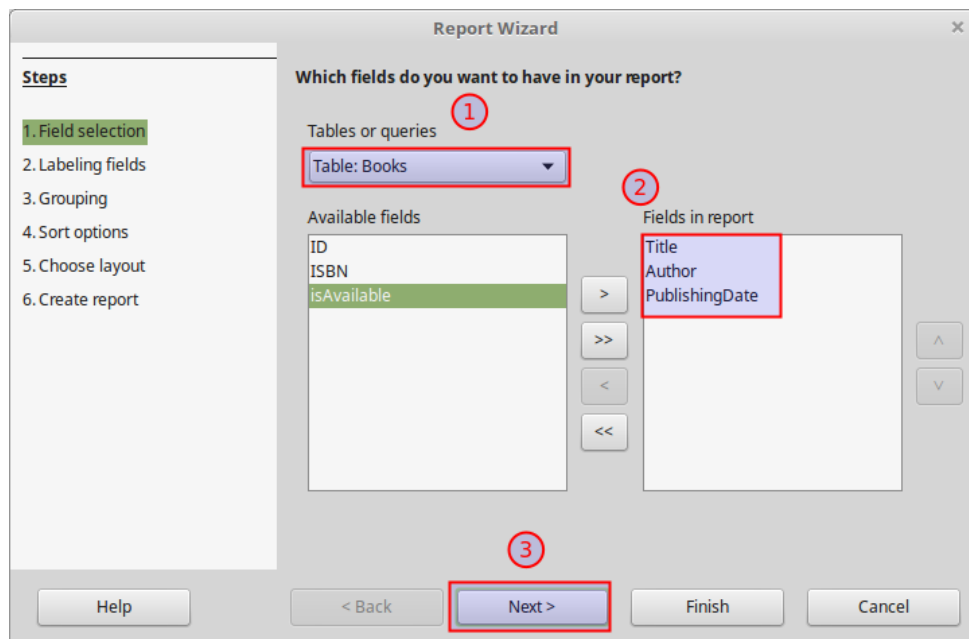
- Create a Report in Design View. Create a report from scratch.
- Use Wizard to Create Report. The Wizard guides you through a step by step process to create the report. Unlike tables and queries, creating a report from scratch can be difficult for beginners. Therefore this is the recommended method for to quickly and easily create a report.

Select the Use Wizard to Create Report. The Wizard opens a dialog window. In the following example we will generate a report that displays all book's title, author and publishing date. The results will be grouped by author.



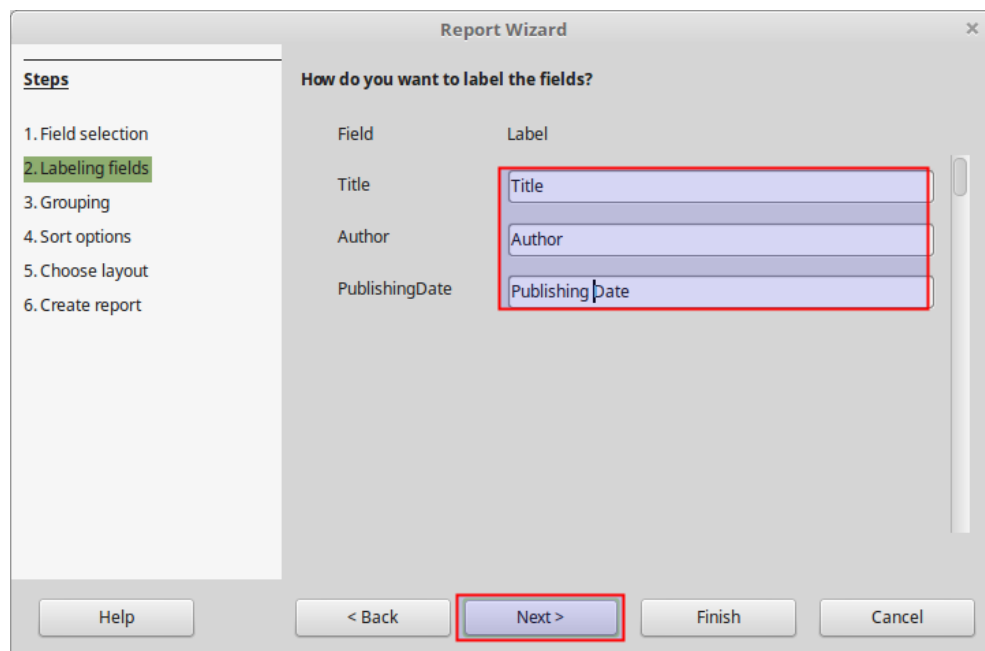
### Step 1

In this step you select the Table Fields that you want the report to contain. In this example we select the Title, Author and PublishingDate fields.



### Step 2

The report generates a label for each field. By default Base chooses the field name for the label name. In our example we change the "PublishingDate" field to "Publishing Date".

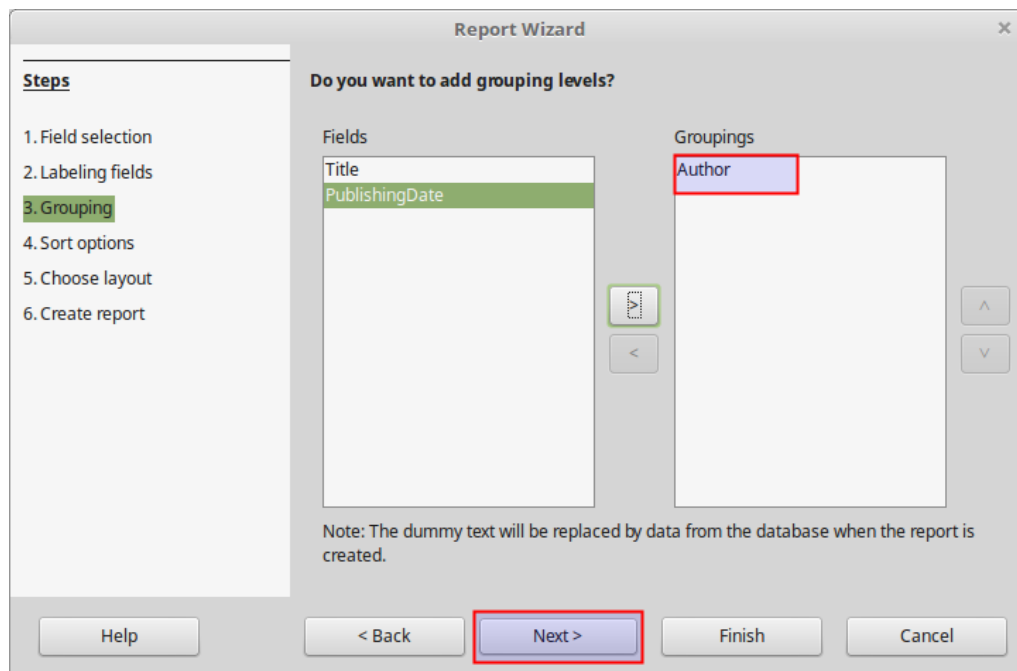


The 'Report Wizard' dialog box is shown at Step 2, 'Labeling fields'. The 'Steps' list on the left includes: 1. Field selection, 2. Labeling fields (highlighted), 3. Grouping, 4. Sort options, 5. Choose layout, and 6. Create report. The main area is titled 'How do you want to label the fields?'. It contains a table with two columns: 'Field' and 'Label'. The 'Field' column lists 'Title', 'Author', and 'PublishingDate'. The 'Label' column contains text boxes with the same labels: 'Title', 'Author', and 'PublishingDate'. A red rectangle highlights the 'Label' column text boxes. At the bottom, there are buttons for 'Help', '< Back', 'Next >' (highlighted with a red rectangle), 'Finish', and 'Cancel'.

Field	Label
Title	Title
Author	Author
PublishingDate	PublishingDate

### Step 3

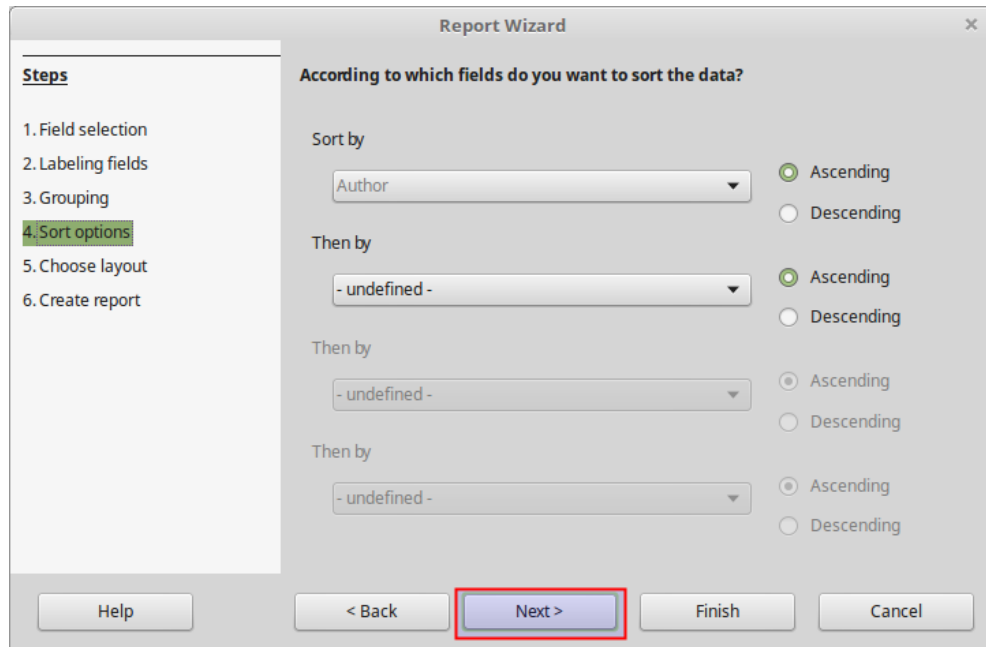
A report can group the results by one or more fields. In our example we choose to group the results by Author.



The 'Report Wizard' dialog box is shown at Step 3, 'Grouping'. The 'Steps' list on the left includes: 1. Field selection, 2. Labeling fields, 3. Grouping (highlighted), 4. Sort options, 5. Choose layout, and 6. Create report. The main area is titled 'Do you want to add grouping levels?'. It contains two lists: 'Fields' and 'Groupings'. The 'Fields' list contains 'Title' and 'PublishingDate', with 'PublishingDate' highlighted. The 'Groupings' list contains 'Author', which is highlighted with a red rectangle. Between the two lists are buttons for adding and removing items. At the bottom, there are buttons for 'Help', '< Back', 'Next >' (highlighted with a red rectangle), 'Finish', and 'Cancel'. A note at the bottom states: 'Note: The dummy text will be replaced by data from the database when the report is created.'

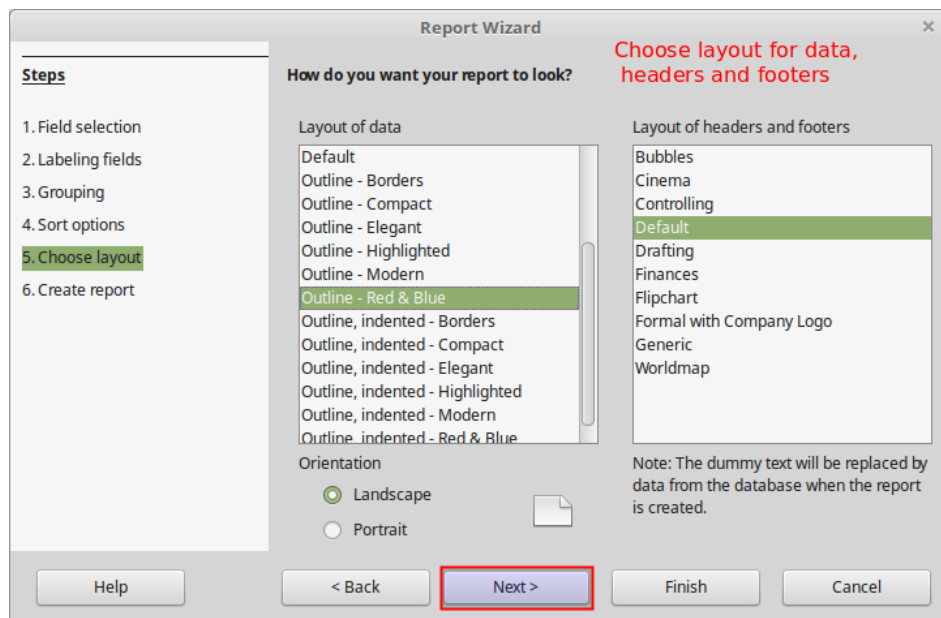
### Step 4

As with queries, the results in reports can be sorted. By default results are sorted by the group field (Author). You can specify more levels of sorting. Leave the default values.



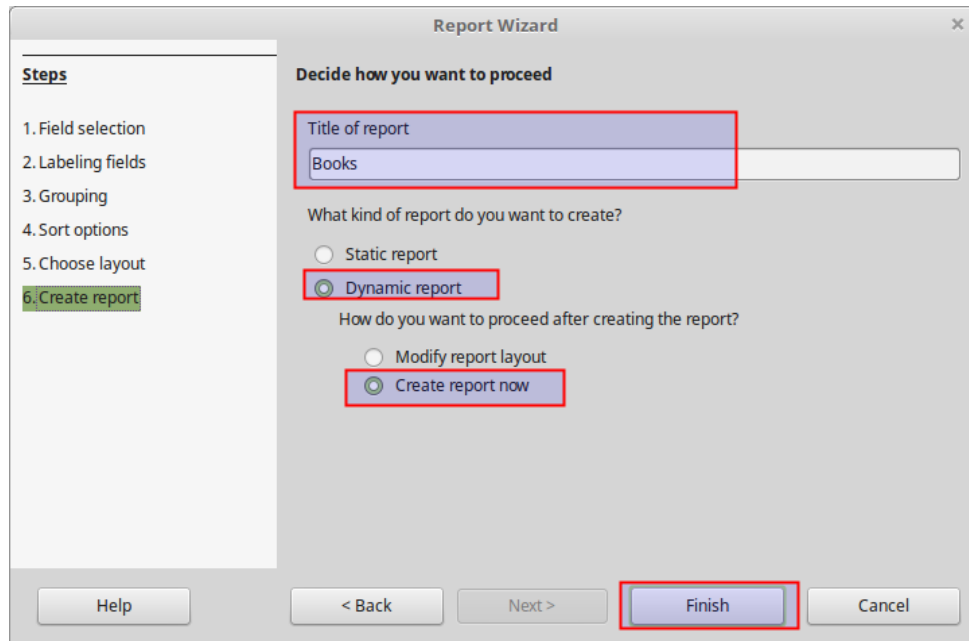
### Step 5

In this step we choose the Layout for the data and the headers and footers of the report. Click on the available options to preview each layout.



### Step 6

Here we specify the title and the type of the report. A dynamic report generates data from the current table data. This means that if we update data on the table that feeds the report, the report will change accordingly.

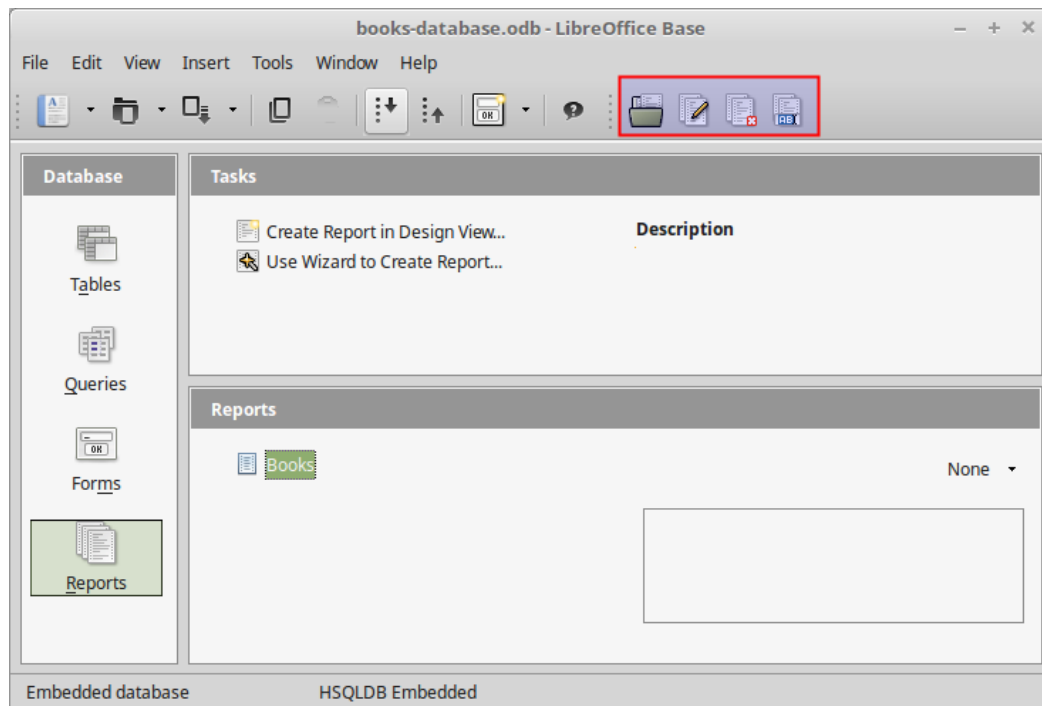


When you finish the wizard the report will be created and opened. The report is a text document and therefore it opens using the Writer component of LibreOffice.

<b>#titleconst#</b>		
#authorconst#	Teo Bous	
#dateconst#	12/25/17	
<b>Author</b>	Franz Kafka	
	<b>Title</b>	<b>Publishing Date</b>
	The Metamorphosis	09/03/1976
<b>Author</b>	Fyodor Dostoyevsky	
	<b>Title</b>	<b>Publishing Date</b>
	The Brothers Karamazov	04/07/2012
<b>Author</b>	George Orwell	
	<b>Title</b>	<b>Publishing Date</b>
	1984	04/12/1979
<b>Author</b>	Homer	
	<b>Title</b>	<b>Publishing Date</b>
	The Odyssey	06/08/1999
	The Illiad	05/07/1998

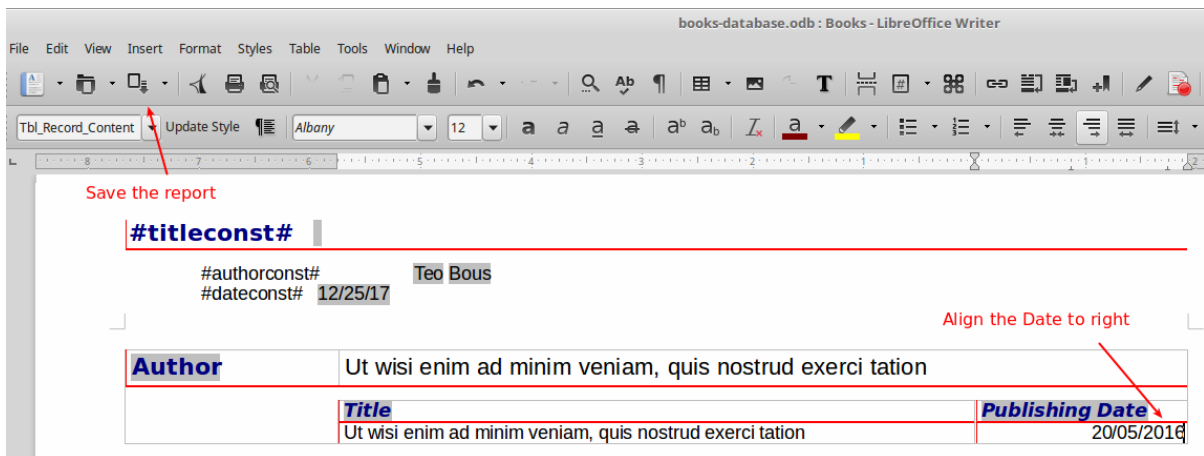
## Managing a Report

To manage a Report first select the Reports Object from the Database Objects Pane and then select the report from the Reports List. Use the Reports toolbar to Open, Edit, Delete or Rename a Report.



### Editing a Report

A Report is basically a Writer document that acquires data from Base. Therefore you can edit a report like any other Writer Document. For example you can change the alignment of the Publishing Date column. When you edit a report you must save it using the Save Button from the Standard toolbar.



### Task 5: Create forms and report for the student Information.



## **Program 6**

### **PASSWORD AND SECURITY FEATURES**

#### **Problem Definition**

Creating password and security features of application.

#### **Problem Description**

All the core of security in the Oracle database user account, without an Oracle username there is no way that one can break into Oracle database. An Oracle database typically has Number of Oracle user accounts that were either created when the database was created or created later by the database administrator. Once created an Oracle user has a number, and privileges that allow him to create and modify database objects.

Once an Oracle user is given a username and password he can connect to Oracle database using any to

#### **Pseudo code**

(1) GRANT Statement Grant privileges to a user (or to a user role)

#### **Syntax:**

Grant System-wide Privs:

```
GRANT system_priv(s) TO grantee
[IDENTIFIED BY password] [WITH ADMIN
OPTION]
```

```
GRANT role TO grantee [IDENTIFIED BY
password] [WITH ADMIN OPTION]
```

```
GRANT ALL PRIVILEGES TO grantee
[IDENTIFIED BY password] [WITH ADMIN
OPTION]
```

#### **Grant privs on specific objects:**

```
GRANT object_priv [(column, column,...)]
ON [schema.]object TO grantee [WITH
GRANT OPTION] [WITH HIERARCHY
OPTION]
```

```
GRANT ALL PRIVILEGES [(column,
column,...)] ON [schema.]object TO grantee
[WITH GRANT OPTION] [WITH
HIERARCHY OPTION]
```

```
GRANT object_priv [(column, column,...)] ON
DIRECTORY directory_name TO grantee
[WITH GRANT OPTION] [WITH
HIERARCHY OPTION]
```

```
GRANT object_priv [(column, column,...)] ON
JAVA [RE]SOURCE [schema.]object TO
grantee [WITH GRANT OPTION] [WITH
HIERARCHY OPTION]
```

grantee: user role PUBLIC

system\_privs:

CREATE SESSION - Allows user to connect to the database

UNLIMITED TABLESPACE - Use an unlimited amount of any tablespace.

SELECT ANY TABLE - Query tables, views, or mviews in any schema

UPDATE ANY TABLE - Update rows in tables and views in any schema

INSERT ANY TABLE - Insert rows into tables

and views in any schema

Also System Admin rights to CREATE, ALTER or DROP: cluster, context, database, link, dimension, directory, index, materialized view, operator, outline, procedure, profile, role, rollback segment, sequence, session, synonym, table, tablespace, trigger, type, user, view.

**object\_privs:**

SELECT, UPDATE, INSERT, DELETE, ALTER, DEBUG, EXECUTE, INDEX, REFERENCES

**roles:**

SYSDBA, SYSOPER, OSDBA, OSOPER, EXP\_FULL\_DATABASE, IMP\_FULL\_DATABASE, SELECT\_CATALOG\_ROLE, EXECUTE\_CATALOG\_ROLE, DELETE\_CATALOG\_ROLE, AQ\_USER\_ROLE, AQ\_ADMINISTRATOR\_ROLE - advanced queuing SNMPAGENT - Enterprise Manager/Intelligent Agent. RECOVERY\_CATALOG\_OWNER - rman HS\_ADMIN\_ROLE - heterogeneous services plus any user defined roles you have available.

*Note: Several Object\_Privs can be assigned in a single GRANT statement*

WITH HIERARCHY OPTION will grant the object privilege on all subobjects, including any created after the GRANT statement is issued.

WITH GRANT OPTION will enable the grantee to grant those object privileges to other users and roles.

"GRANT ALL PRIVILEGES..." may also be written as "GRANT ALL..."

REVOKE Statement Revoke privileges from users or roles. Syntax: Roles: REVOKE role FROM {user, | role, |PUBLIC} System Privs: REVOKE system\_priv(s) FROM {user, | role, |PUBLIC} REVOKE ALL FROM {user, | role, |PUBLIC} Object Privs: REVOKE object\_priv [(column1, column2..)] ON [schema.]object FROM {user, | role, |PUBLIC} [CASCADE CONSTRAINTS] [FORCE]

REVOKE object\_priv [(column1, column2..)] ON [schema.]object FROM {user, | role, |PUBLIC} [CASCADE CONSTRAINTS] [FORCE]

REVOKE object\_priv [(column1, column2..)] ON DIRECTORY directory\_name FROM {user, | role, |PUBLIC} [CASCADE CONSTRAINTS] [FORCE] REVOKE object\_priv [(column1, column2..)] ON JAVA [RE]SOURCE [schema.]object FROM {user, | role, |PUBLIC} [CASCADE CONSTRAINTS] [FORCE] key: object\_privs ALTER, DELETE, EXECUTE, INDEX, INSERT, REFERENCES, SELECT, UPDATE, ALL PRIVILEGES system\_privs ALTER ANY INDEX, BECOME

USER, CREATE TABLE, DROP ANY VIEW  
RESTRICTED SESSION, UNLIMITED  
TABLESPACE, UPDATE ANY TABLE plus  
too many others to list here

roles Standard Oracle roles - SYSDBA,  
SYSOPER, OSDBA, OSOPER,  
EXP\_FULL\_DATABASE,  
IMP\_FULL\_DATABASE plus any user defined  
roles you have available FORCE, will revoke all  
privileges from a user-defined-type and mark  
it's dependent objects INVALID.

### **Drop user**

Drop user username;

**Task:** Create Password and security features for  
employee database

## **Program 7**

### **TABLE LOCKING**

#### **Problem Definition**

Usage of file locking, table locking facilities in application

#### **Problem Description**

Oracle uses lock to control concurrent access to data. Locks are mechanisms intended to prevent destructive interaction between users accessing the same data. Table locks lock the entire tables, while row locks lock just selected rows. Thus locks are used to ensure data integrity while allowing max concurrent access to data by unlimited users.

Locks are used to achieve two important goals.

1. Data concurrency
2. Oracle lock is fully automatic and requires no user action .DBA locks the oracle data while executing SQL statements. This type of locking is called implicit locking. When a lock is put by the user it is called explicit locking.

#### **Types of locks**

Two levels of lock that a DBA can apply. They are

1. Shared : Multi user can hold various share lock on a single resource
2. Exclusive: It prohibits all sharing of resources i.e. only one use has the sole ability to alter the resources until locks are released.

#### **Pseudo code**

```
LOCK TABLE [Table name] IN { ROW  
SHARE | ROW EXCLUSIVE | SHARE  
UPDATE | SHARE | SHARE ROW  
EXCLUSIVE | EXCLUSIVE } MODE [  
NOWAIT]
```

ROW SHARE Row share locks all concurrent access to a table.

SHARE UPDATE They prohibit other users to lock entire table exclusively

ROW EXCLUSIVE Row exclusive locks the same as row share locks, but also prohibit locking in share mode. These locks are acquired when updating, inserting or deleting.

SHARE ROW EXCLUSIVE They are used to lock the whole table to selective update and to allow other users to lock a row in the table but not lock the table in share mode or to update rows.

NO WAIT Indicates that you do not wish to wait if resources are unavailable. All locks are released under the following circumstances:

1. The transaction is committed successfully
2. A rollback is performed
3. A rollback to a save point will release locks set after specified save point
4. Row-level-locks are not released by rolling back to a savepoint
5. Data locks are released by log off

**Task 7:** Demonstrate MySQL locking mechanism for cooperating table access between sessions.