Learn Database Systems with Implementation and Examples

Imed Bouchrika

www.LearnDB.com
# Table of Contents

1 Introduction to Databases 1
   1.1 Motivations ............................................... 1
   1.2 What’s a Database ........................................... 4
   1.3 Database Management System .............................. 5
   1.4 Database Models ............................................. 6
       1.4.1 Hierarchical database model ......................... 6
       1.4.2 Network model ........................................ 8
       1.4.3 Relational model ...................................... 8
       1.4.4 Object-Oriented model ............................... 10
       1.4.5 Object-Relational database ......................... 10
   1.5 Types of Database Systems ............................... 10
   1.6 Database Languages ....................................... 11
   1.7 Phone Book Implementation using Java ................. 11
       1.7.1 JDK & Eclipse Installation ......................... 12
       1.7.2 Java Implementation ................................. 12
   1.8 Questions & Exercises .................................... 16
Bibliography ..................................................... 17

2 Database Design .............................................. 19
   2.1 Database Life Cycle ..................................... 19
   2.2 Planning & Requirement Analysis ........................ 21
       2.2.1 Business Objects ..................................... 22
       2.2.2 Business Rules ....................................... 22
       2.2.3 Users of Data ........................................ 23
       2.2.4 Data Flow Diagram ................................... 24
       2.2.5 User Views ........................................... 24
       2.2.6 Requirements Analysis and Documentation ........ 24
   2.3 DBMS Evaluation and Selection .......................... 25
   2.4 Database Design ........................................... 25
## Contents

2.4.1 Conceptual Database Design ........................................... 26  
2.4.2 Logical Database Design ............................................. 26  
2.4.3 Physical Database Design ............................................ 26  
2.5 Entity-Relationship Diagram ........................................... 27  
2.5.1 Entity, Attribute & Relationship .................................... 27  
2.5.2 Enhanced ER Diagram .................................................. 31  
2.6 Relational Schema Design ................................................. 32  
2.6.1 Entities & Attributes ................................................... 33  
2.6.2 Modeling Relationships ................................................ 33  
2.7 Software for Database Design ........................................... 35  
2.8 Questions & Exercises ..................................................... 36  
Bibliography ........................................................................ 36  

3 SQL : Structured Query Language ........................................ 39  
3.1 Introduction to SQL .......................................................... 39  
3.2 Data Definition Language ................................................ 40  
3.2.1 CREATE ................................................................. 40  
3.2.2 ALTER ................................................................. 43  
3.2.3 DROP ................................................................. 44  
3.3 Data Manipulation Language ............................................. 45  
3.3.1 INSERT ............................................................... 45  
3.3.2 UPDATE .............................................................. 46  
3.3.3 DELETE .............................................................. 46  
3.4 Data Query Language .......................................................... 47  
3.4.1 SELECT .............................................................. 47  
3.4.2 DESCRIBE .......................................................... 53  
3.5 Data Control Language ....................................................... 53  
3.5.1 CREATE USER ...................................................... 53  
3.5.2 GRANT .............................................................. 54  
3.5.3 REVOKE ............................................................ 54  
3.6 Transaction Control Language ........................................... 55  
3.7 Practice SQL ................................................................. 55  
3.7.1 Installing HSQLDB .................................................... 55  
3.7.2 Installing WAMP with phpMyAdmin ............................... 57  
3.8 Questions & Exercises ....................................................... 60  
3.8.1 Part I : SQL Questions ................................................ 60  
3.8.2 Part II : SQL Queries ................................................ 60  
Bibliography ........................................................................ 62

---

4 Advanced SQL Queries
4.1 Aggregate Functions ............................................. 63
4.2 Scalar Functions ................................................. 64
   4.2.1 String Functions ........................................... 64
   4.2.2 Numerical Functions ....................................... 66
   4.2.3 Time Functions ............................................ 66
4.3 GROUP BY .......................................................... 67
4.4 HAVING ............................................................. 68
4.5 Nested Queries .................................................... 68
4.6 Join ................................................................. 69
   4.6.1 Inner Join ..................................................... 70
   4.6.2 Left Join ...................................................... 72
   4.6.3 Right Join ..................................................... 73
   4.6.4 Full Join ...................................................... 73
4.7 SQL Views ........................................................ 75
4.8 Union ............................................................... 76
4.9 Intersect ........................................................... 77
4.10 Foreign Keys ...................................................... 77
4.11 Questions & Exercises ........................................... 80
   4.11.1 Part I : SQL Questions .................................... 80
   4.11.2 Part II : SQL Queries ...................................... 81
Bibliography ............................................................. 83

5 SQL Transactions .................................................... 85
5.1 Introduction ....................................................... 85
5.2 Isolation Levels ................................................... 86
5.3 Transaction Statements ........................................... 89
5.4 Exercises & Questions ............................................ 91
Bibliography ............................................................. 91

6 Procedures & Triggers .............................................. 93
6.1 Procedures ......................................................... 93
   6.1.1 Procedure Variables ....................................... 94
   6.1.2 Procedure Parameters ..................................... 95
   6.1.3 Flow Control ................................................ 97
   6.1.4 Loops ........................................................ 99
   6.1.5 Cursors ...................................................... 101
6.2 Triggers ........................................................... 104
   6.2.1 Creating Triggers in MySQL ................................. 104
   6.2.2 Examples for SQL Triggers ................................ 106
6.3 Exercises & Questions ............................................ 109
## Object Relational Databases

### 7.1 Introduction to ORDBMS

### 7.2 Creating New Types

### 7.3 Handling Objects
- 7.3.1 Object Tables
- 7.3.2 Inserting Objects
- 7.3.3 Querying Objects
- 7.3.4 REF Pointers

### 7.4 Methods
- 7.4.1 Constructor Methods
- 7.4.2 Member Methods
- 7.4.3 Static Methods
- 7.4.4 Map & Order Methods

### 7.5 Inheritance

### 7.6 Collections
- 7.6.1 VARRAY
- 7.6.2 Nested Tables

### 7.7 PL/SQL
- 7.7.1 Variables
- 7.7.2 Flow Control
- 7.7.3 Loops
- 7.7.4 Cursors

### 7.8 Practice Oracle

### 7.9 Questions & Exercises

### Bibliography

## Databases & Java Programming

### 8.1 MySQL with Java
- 8.1.1 Inserting Contacts
- 8.1.2 Searching Contacts
- 8.1.3 Running the Java Application

### 8.2 HSQLDB

### 8.3 Oracle with Java

### Bibliography
Chapter 1

Introduction to Databases

1.1 Motivations

Before we define or explain what a database or even argue whether it is important for software development, let us ask the simple question: Can we create a software application without the use of a database system. To make it easier for you to answer such question, we take the simple example for an online phone directory which can be developed using a web scripting programming language such as PHP. The application is normally used by online visitors to search for phone numbers by typing the person surname as well as the address. Figure (1.1) shows an example for the online phone directory provided by British Telecom to search for people phone numbers within the United Kingdom.

The question that poses is where and how we can store the list of people’s names and their phone numbers permanently as well as retrieve them easily. In other words, how and where to store data such that it can be accessed by the web application. Trivially, the data would never be stored inside the application code as any modification needed for the data requires altering the code and therefore exacerbating compilation and deployment
Figure 1.2: Architecture for Phone Directory using File-Based Storage.

again which is an expensive and cumbersome process. The simplest solution would be to store data inside a normal text file whereby we format the data using a simple pre-defined structure. For instance, we store each record for a new person on a new line whilst the values for a record including name, surname and phone number should separated by a comma “,”. That’s what we call a CSV file “comma separated values” as illustrated in Figure (1.2) which shows a simple architecture for the phone directory. There is a web interface written usually in HTML having a simple form to search for a person by name. Upon clicking the “search button” within the form, data is transmitted into the web server for processing. The server handles search and retrieval of data from the csv text file.

Listing 1.1: Simple PHP Code for a Phone Directory

```php
<?php
function searchDirectory($name){
    $results=array();
    $people=file("mydata.csv");
    for($i=0;$i<count($people);$i++){
        $person=explode(","(strtolower($people[$i]));
        if(strpos($person[0],$name)!==FALSE)
            $results[]=$person;
    }
    return $results;
}
?>
```

Listing (1) shows a simple code written in PHP programming language for a function that searches a text file named mydata.csv for a person record by a given name. The code contains less than 10 lines which is enough for a basic phone directory. The aim of this code is point to the simplicity of accessing and searching data files for developing the
application. The lines of codes are explained as follows:

• Line 2: declaration of a function named `searchDirectory` that takes one parameter named `name` which is submitted through the HTML form.

• Line 3: Initialisation of an array named `results` for storing results.

• Line 4: Opening the csv file `mydata.csv` into an array named `people`. Every line i.e. record from the file is stored as an element within the array.

• Line 5: `For Loop` to iterate through all records within the csv file.

• Line 6: Splitting a single line which is a person record, into separate elements which are stored into an array named `person`.

• Line 7: Comparing if the string `name` is contained with the first name of the person

• Line 8: There is a match to be added inside the return results.

From the example being illustrated earlier, it becomes obvious that software applications can be developed without a fancy or advanced database system through the use of simple textual-based files. However, a number of basic questions and issues may arise when using such approach. For instance:

• What if we want to enable concurrent or distributed management by different users connecting from remote different locations to the storage facility or files?

• What about if we want to grant different access rights or roles to different users. For example, we grant read-only for guests and read-write for administrators.

• What about the scalability and availability of the system when adding over a million of records where the performance of the search suffers greatly as it would scan a large number of lines within the file.

Further and critical issues and challenges that needs to be addressed for software applications using flat-files for storage includes: File management, concurrency and multiple access, advanced search or operations, scalability, adding new functionalities, integrity checking, maintenance and backup and security.

Back to the first question of whether we can develop a software application without the use of an advanced database system. Yes, it can be done easily as shown earlier, but life would `difficult and tough` to address a large number of issues and functionalities for the system. This signifies the importance of using database systems and their critical role for software application as they were invented primarily to address the limitations encountered when using files as a direct storage facility. To further motivate you to learn databases in terms of money and job potentials, a comparative analysis for the average
salary for different job roles within the United States of America, have showed that the average salary for an Oracle Database Administrator exceeds the average salary for Java Programmer, Research fellow and even a dentist as shown in Figure (1.3). The job role for a database administrator is usually to ensure the smooth functioning of an existing database system including maintenance and security hardening for the system. Other database related job roles include database designer, analyst and developer.

![Figure 1.3: Comparative Analysis for Average Salary within the United States of America](Source: www.indeed.com, 2013]

1.2 What’s a Database

In plain English, a database is defined as a collection or repository for storing data or information which are:

- **Interrelated**: We mean by interrelated as parts within the database has a relation or an association to other parts of the data within the database. For instance, we create a database for a university to store the list of courses as well as enrolled students. The students must have a relationship to courses.

- **Organized**: The data is usually arranged depending on the database model and structure as well as the requirements for the application. Organization of data includes grouping objects with the same attributes and properties into the same storage entities whilst creating relationship among different entities to relate them to each other.

- **Accessible & Exploitable**: The database should provide access protocols for third party applications to retrieve and store data using different programming language such as Java as well as different platforms including mobile and desktop systems.
The definition from the Oxford dictionary as:

“Database: an organised body of related information.”

Databases are used on a daily basis for a variety of reasons including e-commerce, GPS and maps navigation, banking and so on. With a database, users or companies should be able to store data in an organized manner. Once the data is stored, it should be easy to retrieve such data efficiently and swiftly.

1.3 Database Management System

The database management system (DBMS) is a software system provided to ease the management and functioning of databases including definition, creating, querying, updating and administration of database. The DBMS can be usually installed and accessed by the database administrator through the console, web interface or simple graphical user interface depending on the type of the DBMS. In addition, DBMS vendors provide various drivers for different programming languages and platforms to interact with the database engine. The main functionalities for the database management system are listed as follow:

- **Data Manipulation**: This includes the alteration of stored data as well as the search and retrieval of information. The Database management system usually handles the processes and the structure of the data on the hard disk.

- **Data Sharing**: As a major property for database system is to allow concurrent access by multiple users at the same time to share data, the database management system provides the necessary mechanism for concurrency control through the use of transaction management such as locking, two-phase locking and time-stamping to avoid conflicts and deadlocks.

- **Data Backup and Recovery**: In the event a catastrophe occurs DBMS must provide ways to recover a database so that data is not permanently lost. The easiest way to do this is to make regular backups of information, employ journalling, set-up Master-Slave mode or utilize other advanced options

- **Security and Privacy**: Security is the prevention of unauthorized users accessing the database or part of the database. The DBMS uses the following concepts to ensure the security for the database:
  
  - **Encryption**: is when DBMS converts the data in a database to an indecipherable format.
  
  - **Authentication**: is a technique in which the database administrator can identify the person accessing the database. Usually using a **Username** and **Password**.
- **Authorization**: is a set of rules that database administrators (DBA) set up to specify the levels of usage that individuals or groups are allowed to have.

- **Data Integrity**: This is a set of rules that DBMS provides to see that data integrity is enforced, thus avoiding incorrect or inconsistent data. Integrity rules include:
  - Data types
  - Legal values and Format
  - Key Uniqueness and referential integrity.

There are a large number of DBMS on the market today from open source application to proprietary enterprise database management system such as Oracle. Major examples for the DBMS summarized in the following table:

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Vendor</th>
<th>License</th>
<th>Features/URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>Oracle Inc.</td>
<td>Proprietary</td>
<td>Object-Relational</td>
</tr>
<tr>
<td>MySQL</td>
<td>Oracle Inc.</td>
<td>Open Source</td>
<td>Relational</td>
</tr>
<tr>
<td>MS SQL</td>
<td>Microsoft</td>
<td>Proprietary</td>
<td>Relational</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>PostgreSQL</td>
<td>Proprietary</td>
<td>Object-Relational</td>
</tr>
<tr>
<td>MongoDB</td>
<td>MongoDB</td>
<td>Open Source</td>
<td>NoSQL</td>
</tr>
<tr>
<td>Access</td>
<td>Microsoft</td>
<td>Proprietary</td>
<td>Relational</td>
</tr>
<tr>
<td>SQLite</td>
<td>-</td>
<td>Open Source</td>
<td>Embedded Relational</td>
</tr>
<tr>
<td>Sybase ASE</td>
<td>Sybase</td>
<td>Proprietary</td>
<td>Relational</td>
</tr>
<tr>
<td>Firebird</td>
<td>-</td>
<td>Open Source</td>
<td>Relational</td>
</tr>
<tr>
<td>MarkLogic</td>
<td>MarkLogic</td>
<td>Proprietary</td>
<td>NoSQL</td>
</tr>
<tr>
<td>BaseX</td>
<td>-</td>
<td>Open Source</td>
<td>XML Native</td>
</tr>
</tbody>
</table>

### 1.4 Database Models

A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated. In short: Database models are concepts used for structuring the defining data for storage. Popular Database Models include hierarchical, network, relational and object-relational. Figure (1.4) previews the lifespan for the different models over the years.

#### 1.4.1 Hierarchical database model

It is a data model in which the data is structured logically into a family tree. The structure allows the representation of data via parent-child relationships: each parent can have many children, but each child has only one parent attached to it. All attributes of a specific
Parent-Child Relationship Type is basically $1:N$ relationship meanwhile the schema for a hierarchy has one single root. This model is recognized as the first database model created by IBM in the 1960s. Nowadays, the most widely used hierarchical database systems are the Information Management System (IMS) of IBM along with the Windows Registry by Microsoft. Figure (1.5) shows an example for the use of hierarchical model where we store information about students, departments and courses within a university. For every entity, there is a list of attributes associated with it as the department has name and number. The department and student data forms a hierarchy, where the department data represents the parent segment and the student or course data represents the child segment.
1.4.2 Network model

The popularity of the network data model had coincided with the popularity of the hierarchical model when it first emerged. As some data were more naturally modelled with more than one parent per child, the network model is introduced for the modelling of many-to-many relationships in data as opposed to the hierarchical model which supports only 1:N relationship. In 1971, the Conference on Data Systems Languages (CODASYL) formally defined the network model. The CODASYL network model is based on mathematical set theory. Figure (1.6) shows the previous example for the university using the network model where the different entities are organised in a graph, in which some entities can be accessed through several path.

![Network database model](image)

Figure 1.6: Network database model

1.4.3 Relational model

The relational model for databases was developed by E.F. Codd which allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organised in tables. The table is a collection of records and each record in a table contains the same fields. The main features for relational tables include:

- Values should be atomic
- Each row is unique
- Column values are of the same data type
- The sequence of columns of a table is insignificant
- The sequence of rows should be considered insignificant
- Each column has a unique name (attribute).

Certain columns may be assigned as keys such that searches for specific values of that field will use indexing to speed them up. Furthermore, the same keys on different tables can be used for joining or relating data from the different tables by matching values in those fields. The relational database model is based on the Relational Algebra. The university example
is depicted in Figure (1.7) using the relational model where each entity is drawn as a table. Attribute for the entity is shown as a column within the table meanwhile relationship can be expressed using simple attributes within a table or instead by a set of attributes on its own table.

![Relational database model](image)

Figure 1.7: Relational database model

The relational model is today the primary data model for commercial data-based applications. It attained its primary position because of its simplicity and ease of programming integration using various programming languages compared to earlier data models such as the network model, the hierarchical model or even recent database models of which their complexity can be argued. Aside from the relational model being the most prevalent of all the database models, other important reasons can include the followings:

- Definition, maintenance as well as manipulation of data storage structures are practically easy.
- Data is retrieved through simple *adhoc* queries.
- Data is well protected.
- Well-established ANSI (American National Standards Institute) and ISO (International Organization for Standardization) standards exist.
- Many vendors offer a plethora of products.
- Conversion between vendor implementations is relatively easy through the use of SQL
- Relational DBMSs are mature and stable products.
1.4.4 Object-Oriented model

Object or Object-Oriented database system are enriched with functionalities of object programming languages including classes, encapsulation and inheritance. Object DBMSs extend the semantics of the C++, Smalltalk and Java object programming languages to provide full-featured database programming capability for storing persistently objects, while retaining native language compatibility. In contrast to a relational DBMS where a complex data is hardly stored and related to each other as well as the fact that the retrieval for complex data is even worse, as data must be flattened out or serialized to fit into tables or joined together from those tables to form the in-memory structure. Object DBMSs have no performance overhead to store or retrieve a hierarchy of interrelated objects. The primary benefit of this approach is the unification and integration of the application and database development into a seamless data model and language environment. As a result, applications require less and clear code, use more natural data modelling, and developed software is easier to maintain. Object developers can write complete database applications with a modest amount of additional effort.

1.4.5 Object-Relational database

Object-Relational database management systems (ORDBMSs) add new object storage capabilities to the relational systems at the core of modern information systems. These new facilities integrate management of traditional fielded data, complex objects such as time-series and geospatial data and diverse binary media such as audio, video, images, and applets. By encapsulating methods with data structures, an ORDBMS server can execute complex analytical and data manipulation operations to search and transform multimedia and other complex objects. Oracle is the major ORDBMS being heavily used for enterprise application.

1.5 Types of Database Systems

As there are different database models for the structuring of data, databases can be classified further into different categories depending on different factors such as the type of data being stored for instance, bibliographic, document-text, statistical, or video-based. Further categorisation of databases is based on the technical aspect or technology being adopted for the database. We list in this section a few of types of databases with different objectives.

- **In-memory** database resides in main working memory, but is typically backed-up into non-volatile persistent data storage.

- **Cloud** database relies on cloud technology. Both the database and most of its DBMS reside remotely. An example of cloud storage is Amazon S3.
• **Deductive** database combines logic programming with a relational database.

• **Distributed** database is one in which both the data and the DBMS are distributed over multiple different computers.

• **Document-oriented** database is designed for storing, retrieving, and managing document-oriented, or semi-structured data, information. Document-oriented databases are one of the main categories of NoSQL databases. NoSQL is an approach to data management and database design using optimized *key-value* stores which is useful for simple storage and fast retrieval of very large sets of distributed data.

• **Embedded** database system is a DBMS which is tightly integrated with an application software that requires access to stored data in such a way that the DBMS is hidden from the application’s end-users. SQLite and HSQLDB are well-known examples that fall under this type.

• **Federated** database system comprises of several distinct databases where each has its own small DBMS. There is a single Federated Database Management System (FDBMS) that handles the coordination and management of the smaller DBMSs.

### 1.6 Database Languages

Database languages are specific to a particular data model or database type. Notable examples include:

• **SQL**: Structured Query Language mostly used for Relational Databases. We will show the use of SQL in Chapter 3.

• **OQL** is an object model language standard (from the Object Data Management Group).

• **XQuery** or **XPath** are standard XML query languages implemented by XML database systems such as MarkLogic and eXist.

• **SQL/XML** combines XQuery with SQL.

### 1.7 Phone Book Implementation using Java

In this tutorial, we will create a simple phonebook system using a *csv* text file with Java Programming Language. The application will be implemented without accessing or using any advanced database management system like Access, MySQL or Oracle. You are initially required to install the Java Development Kit (JDK) as well as Eclipse as an Editor for writing the Java code.
1.7.1 JDK & Eclipse Installation

Java Development Kit (JDK) is the software needed to compile and execute software applications written in Java. The JDK comes as a bundle of software that you can install on your computer to develop Java based software. Meanwhile, the “JRE” is the Java Runtime Environment that is the implementation of the Java Virtual Machine which actually executes Java programs. Typically, each JDK contains one JRE's along with the various development tools like the Java source compilers, bundling and deployment tools, development libraries, etc. You can download the latest version of JDK from the official Oracle.com website using the link below:

http://www.oracle.com/technetwork/java/javase/downloads

Once you have installed the JDK, You need to download an Integrated Development Editor (IDE) such as Eclipse for Java Developers. The Eclipse IDE contains what you need to build and compile Java applications. Considered by many to be the best Java development tool available. Eclipse IDE for Java Developers provides superior Java editing with validation, incremental compilation as well as code assistance such as code auto-complete. Eclipse IDE for Java Developers can be downloaded from the address below:

http://www.eclipse.org/downloads

Figure (1.8) shows a screenshot for the Eclipse IDE for Java Developers installed on a Windows operating system:

1.7.2 Java Implementation

The program will be a textual-based interactive application without a graphical interface for the sake of simplicity! We will be having three main sections for writing the Java application for the PhoneBook:

- Interaction with the user.
- Adding Contacts
- Searching for a contact.

In case you are using Eclipse, you may need to create a new Java Project with any given name. Afterwards, Create a new Class through Clicking File $\rightarrow$ New $\rightarrow$ Class. You need to name the class as PhoneBook. Copy the following code for the PhoneBook Class:
Listing 1.2: PhoneBook.java: Java Code Structure for the PhoneBook.

```java
import java.io.*;
import java.util.*;

public class PhoneBook {
    public static String help_msg= "Press: H Help - A Add contact - S Search - Q Exit :";
    public static void main(String[] args) {
        System.out.println("**** Welcome to MyPhone Book ****
        Scanner s=new Scanner(System.in);
        for(;;){
            System.out.println("[Main Menu] "+help_msg+"\n: ");
            String command=s.nextLine().trim();
            if (command.equalsIgnoreCase("H")){
                System.out.println(help_msg);
            }else if (command.equalsIgnoreCase("A")){
                System.out.print("Type in contact details in the format: name,lastname,phone\n: ");
            }else if (command.equalsIgnoreCase("S")){
                System.out.print("Type in the name you are searching for :\n: ");
            }else if (command.equalsIgnoreCase("Q")){
            }
        }
    }
}
```

Figure 1.8: Eclipse IDE for Java Developers
19 |     System.out.println("Good Bye User....");
20 |     System.exit(0);
21 | }else{
22 |     System.out.print("Unknown command ! Try again 
: ");
23 | }
24 | }
25 | }
26 | }

- **Line 5**: `public static String help_msg = ...`
  We are declaring a variable as static => The variable belongs to the class. Accessing static variables through the format: `ClassName.VariableName`

- **Line 8**: `Scanner s = new Scanner(System.in);`
  Scanner is a special class for reading from the keyboard when taking the special object `System.in` as a parameter.

- **Line 11**: `s.next().trim();`
  `s.next()` is to read the next input from the keyboard, directly followed by `trim()` to remove (clip) special characters like space, new line from the start and end of the word.

The part of the code for adding a new record is listed below within the case where the user presses the key A.

```
Listing 1.3: Java Code for Adding a New Person.

1 |     System.out.println(help_msg);
2 | }else if (command.equalsIgnoreCase("A")){
3 |     System.out.print("Type in contact details in the format: name,lastname,phone\n: ");
4 |     String line = s.nextLine().trim();
5 |     String[] tmp = line.split(",");
6 |     while (tmp.length != 3){
7 |         System.out.print("Error, Try the format : name,lname,phone\n: ");
8 |         line = s.nextLine().trim();
9 |         tmp = line.split(",");
10 |     }
11 |     try{
12 |         FileWriter fs = new FileWriter("phonebook.csv", true);
13 |         BufferedWriter out = new BufferedWriter(fs);
14 |         out.write(line+"\n");
15 |         out.close();
16 |     }catch(Exception e){e.printStackTrace();}
17 | }
18 | ...
```

• **Line 18, line.split(“,”)**: This breaks or split the string using the separator “,” into a set of elements stored within an array. If you run this program, under your project workspace, you will see a new file being created called `phonebook.csv`, try to open it with Excel or any simple text editor.

In order to search for a contact by name, we ask the user to type in the name of the person to search for. The program would load up the csv file and search for the person’s record line by line. Whenever a match is found when comparing, the record is printed on the screen to the user.

**Listing 1.4: Java Code for Searching for a Person.**

```java
... } else if (command.equalsIgnoreCase("S")){
    System.out.print("Type in the name you are searching for :\n:");
    String search=s.nextLine().trim();
    search=search.toLowerCase();
    try{
        File f=new File("phonebook.csv");
        Scanner s2=new Scanner(f);
        while(s2.hasNextLine()){
            String line=s2.nextLine();
            if (line.toLowerCase().indexOf(search)>=0){
                System.out.print("\nResult: "+line);
            }
        }
        System.out.print("\n\n");
    } catch(Exception e){e.printStackTrace();}
} ...
```

- **Line 34: s.toLowerCase()**
  Convert a string to small letters.

- **Line 37: Scanner s2=new Scanner(f)**
  This is a Scanner object for reading the content of file object `f`.

- **Line 40: s.indexOf("search")**
  returns the position of the term “search” if it is contained within the string `s`. If not found, it would return instead -1.
1.8 Questions & Exercises

Try to have a go on the following questions:

1. what’s a database ?

2. What’s the difference between data and database ?

3. What’s the difference between database and DBMS ?

4. List the main challenges for using file-based storage.

5. List the main benefits of using a DBMS.

6. Mention if any, the disadvantages of DBMSs

7. What do we mean by data integrity ?

8. What’s the main difference between the hierarchical and network database models ?

9. What is the main advantage of the relational model versus other models?

10. Explain the concepts for the DBMS for handling security

11. What’s the difference between database type and model ?

12. List two common tasks that a DBA has to perform

13. List the different job roles related to databases.

14. What do we mean by Serializing complex data ?

15. What’s the difference between SQL and OQL ?

16. What are the properties of a relational table ?

17. What’s the difference between in-memory and embedded databases ?

18. List at least three XML-based DBMSs ?

19. Explain the notion for NoSQL.

20. Try to extend the Java code for the PhoneBook to support deleting existing records.
Bibliography


Chapter 2

Database Design

2.1 Database Life Cycle

The software development can be defined as a set of activities required to efficiently transform the user’s need into an effective software solution. The life cycle of software systems differs from product to product going through various phases. The software development lifecycle (SDLC) typically includes the following phases: planning & management, requirements analysis, design, implementation, testing & quality assurance and operation and maintenance phase. A database is usually a fundamental component of the software application, especially in business oriented systems. Thus database design is part of system development. In the same way, the database development life cycle (DDLC) which is a

![Figure 2.1: Phases for the Database Development Life Cycle.](image-url)
part of the software development life cycle, has its own various phases which are listed in Figure (2.1). The different phases for the database development are outlined briefly below, meanwhile the following sections explain in details the important phases of database development.

1. **Requirements Collection and Analysis**
   - During the planning and requirements analysis phase, the collection and analysis of the information regarding the part of the enterprise to be served by the database are completed including the description of the data to be used or generated.

2. **Evaluation and Selection**
   - In this phase, the criteria for the database management system (DBMS) are defined and evaluated to choose a product such as Access, SQL Server, MySQL, Oracle.

3. **Database Design**
   - The database design phase is divided into three steps: conceptual database design, logical database design and physical database design. In the conceptual database design phase, the model of the data to be used independent of all physical considerations is to be constructed. The model is based on the requirements specification of the system. In the logical database design phase, the model of the data to be used is based on a specific data model, but independent of a particular database management system is constructed. This is based on the target data model for the database e.g. relational data model. In the physical database design phase, the description of the implementation of the database on secondary storage is created.

4. **Implementation**
   - In most databases, a new database implementation requires the creation of special storage related constructs to house the end user tables. These constructs usually include storage group, tablespaces, data files, tables and so on.

5. **Data Migration & Loading**
   - After creating the database, the data must be loaded into the database. If the data to be loaded into the database is currently stored in a different system or in a different format, then the data needs to be converted and then migrated to the new database. Data conversion and migration tools and utilities are available with almost all database management systems available in the marketplace.

6. **Testing and Performance Tuning**
   - Once the data is loaded into the database the database is tested and fine-tuned for performance, integrity, concurrent access and security constraints. The testing and performance tuning occurs in parallel with the testing and performance tuning of the application programs. Sometimes, the performance degradation of the database is due to the inefficient code in the application program.

7. **Operation**
   - Once the data is loaded to the database and it is tested, the database is released into production (along with the application programs). At this stage the database is considered to be operational and the database, its management, its users and the application pro-
grams together form an information system. During the operational phase, the database is accessed by the users and application programs, new data is added, the existing data is modified and some obsolete data is deleted. The database administrators perform the administrative tasks like performance tuning, storage space creation, access control, database back up and so on. It is during the operational phase that the database delivers its usefulness as a critical tool in management decision-making and help in the smooth and efficient functioning of the organization.

8. Maintenance

Once the database is released into production, it will not remain as it was designed, new business requirements, need for new information, acquisition of new data and similar factors will make it necessary to make modifications and enhancements to the existing design. So the database administrators will definitely receive requests for more storage space, changes in the database design, addition of tables, addition of new users, removal of users who have left the organization, changes in the access privileges of the users and so on. The main tasks in this phase are:

- Database backup and recovery.
- Performance tuning of the database.
- Database design modifications.
- Database access and users management.
- Database audits including access and security audits.
- Hardware and Server maintenance and upgrade.

2.2 Planning & Requirement Analysis

The database planning phase includes the activities that allow the following stages of the database system development lifecycle to be realized as efficiently and effectively as possible. This phase must be integrated with the overall information system policy of the organization. The very first step in database planning is to define the mission statement as well as objectives for the database system. That is the definition of:

→ The major aims of the database system.
→ The supported tasks of the database system.
→ The resources of the database system.

The first step in implementing a database system is to find out what is required. What kind of a database is needed for the organization, what is the volume of the data that must be handled, how much data is to be stored and so on. This phase is called the Requirements Analysis. Requirements analysis is the first and most important stage for
the database development life cycle. It is also the most labor-intensive for the database
designer. This stage involves assessing the informational needs of an organization so that
a database can be designed to meet those needs. The overall purpose of requirements
analysis is to gather every bit of information needed to design a database that meets the
informational needs of the company. This is accomplished by performing a series of related
tasks including conducting interviews with end-users and managers, examining existing
databases and systems. It is critical for the designer to approach requirements analysis
armed with a plan for each task in the process.

2.2.1 Business Objects

The process of identifying business objects is typically the first phase during the require-
ment analysis. Business objects are things which have properties or characteristics within
the business environment that are related. Business objects can be tangible or non-tangible
entities that can be stored with their own properties such as employees, projects, cus-
tomers, appointments, products, orders, suppliers. During requirements analysis, business
objects must be identified and sorted according to their class or category. Keep in mind
that every organization has its own policy and decisions about which business objects it will
and will not store data about. For example, for a small business, the owners might decide
they do not want to have any data about their everyday clients stored on the company
database.

Business objects are usually converted into entities later on during the design phase. Enti-
ties, in turn, are ultimately translated into database tables with SQL during the im-
plementation phase. Every business object has characteristics that describe it. Customers,
employees and suppliers, for example, will have names, addresses, phone numbers, and so
on. Products will have names, prices, descriptions, and so on. Appointments will have
times, dates, and so on. In fact, if you cannot identify two or more characteristics of a
business object, there’s a strong possibility it’s not a business object at all. The charac-
teristics of business objects are converted into the attributes of entities in the conceptual
and logical design stages. Attributes, in turn, are ultimately translated into table fields
with SQL. Business objects relate to each other in some form or fashion. Customers place
orders, employees take orders, orders are for products, products come from suppliers...

2.2.2 Business Rules

Business rules describe the business policies and decisions that apply to the data stored on
an organization databases. Business rules reflect how a business perceives its use of data.
As in fact that some business rules are especially essential to the database designer because
they can influence the design for the logical schema of the database. The business rules are
usually ensured or honored by the designer through the use of certain constraints that can
be applied on a database. These constraints help to preserve data integrity. Business-rule
constraints fall into two categories:

1. **Field constraints within tables**: There are various field constraints that can be imposed on a database to honor business rules. For instance, let us consider the following business rule for shipping policy where there is a restriction to ship products only to four countries: Austria, Germany, Belgium and Spain.
   Field constraint: These four countries are represented in the table for storing customers as a field or attribute called Country. A field constraint is placed on the Country field so that only those four countries are accepted and stored into the database for that specific table.

2. **Relationship constraints between tables**: There are different constraints that can be placed on the relationships between entities. Consider the following business rule: Every customer must have at least one valid paid order.
   Relationship constraint: The relationship between the customer table and orders table must be governed by a participation constraint in which case every record from the orders table must be related to at least one record from the clients table.

### 2.2.3 Users of Data

The next step in the process of Requirements Analysis after documenting and analyzing existing business objects and their rules, is to interview people within the company who are going to actually use the data in the database. As this can prove to be an important task to either revise the collected requirements or add new requirements. The users of the database can include employees and managers. In short, there are three major reasons why users of data are interviewed by database designers:

- To find out whether users require additional information.
- To find how data are currently being used.
- To determine future growth requirements to account for scalability or new potential features.

For instance, an initial business rule is documented that shipping policy is restricted to only four countries. However, after interviewing the company manager reveals that they are intending to sell their products to all countries within Europe in about 20 months, the designer might refrain from placing any field constraints on country-related fields that would preclude data entry about overseas customers. Again, documentation of the interviews is extremely important.

---

2.2.4 Data Flow Diagram

As part of Requirements Analysis phase, the designer should preferably request or prepare organizational charts from the company, and supplement these charts with information gathered during interviews with users to assess how data flow is handled within the company. They use this information to create a data flow diagram. These diagrams are useful to the designer in establishing user views across various databases as well as for creating an efficient user interface for the application. It is not always necessary to create a data flow diagram. However, The larger the organization, the more important it is to create a data flow diagram as they help application designers in planning database application programs.

2.2.5 User Views

User views are specific views of data created with SQL which are further explained in the following chapters. The designer defines what data should be made available using a set of stored queries. There are three important reasons for creating user views:

- Restrict or limit access to the data such that a user can see or even update what they need and no more.
- Summarize data from various tables which can be useful to generating reports as well as generating new fields.

User views, therefore, not only protect sensitive data within an organization; they also protect users from being inundated with useless data.

2.2.6 Requirements Analysis and Documentation

Importantly, all tasks and activities accomplished during the Requirements Analysis phase must be carefully documented as such documents would carry over to the next processes with the database development life cycle. The following documents should be produced during this phase:

1. Business objects, their characteristics, and how they relate to one another
2. Business rules including possible constraints.
4. Specification list for user views.
2.3 DBMS Evaluation and Selection

Just to note that the phase for the DBMS selection is sometimes classified after the design process, but we would argue that DBMS may influence the design phase and therefore should come first. During this phase, we evaluate the different database management systems and select the one that is ideally suited for the needs of the organization. The main factors that influence the selection of the DBMS are:

- **Cost of the System**: The cost includes the purchase price, cost of operation, maintenance, site license, installation, training, data migration, data conversion, etc.

- **Features and Tools**: Not all database management systems are created equally with the same features. Some systems have more features than others. Some have a lot of data administration, querying and report writing tools as part of the system. For example, report generators, query generators, data loaders, data dictionaries and so on make the DBMS easy-to-use and pleasant to work with. Similarly DBA utilities, automated back-up/recovery systems, access control and management systems all make the DBMS more attractive to the buyer.

- **Customer Support and Training**: Another factor that will influence the selection of the DBMS is the efficiency of the customer service (after sales service) that the DBMS vendor offers. Also the ease of training, that is the ease with which users can be trained in the system is another factor.

- **Underlying Data Model**: The purchasing decision is to a very large extent influenced by the underlying data model—Hierarchical, Network, Relational, Object-Relational and so on.

- **Portability**: The DBMS selected should be portable across platforms and languages if there is a requirement for that.

- **Hardware Requirements**: The hardware requirements of the DBMS is another important factor. If the DBMS needs high-end systems to perform efficiently, then the cost of these hardware components should also be considered while making the selection for the DBMS.

2.4 Database Design

In this phase, the database designers will decide on the database model that is ideally suited for the organization's needs. The database designers will study the documents prepared by the analysts in the requirements analysis phase and then will go about developing a system that satisfies the requirements. The database design phase is divided into three steps.
2.4.1 Conceptual Database Design

At first, a conceptual design of the database is created. In the conceptual design stage, data modeling is used to create an abstract database structure that represents the real-world scenario. The conceptual model will be a true representation of the real world, only if the requirements analysis is properly done, as it needs a thorough understanding of the business and functional areas. At this stage the hardware or the database model that is to be used are not decided—the conceptual design is hardware and software independent.

The database designer creates a data model of the system. The business contains entities and relationships. Each entity will have attributes. In this step the business entities and relationships are transformed into a data model (usually an E-R model) using E-R diagrams. Now many designers have started using data modeling using UML (Unified Modeling Language) instead of the E-R diagrams. Once the data model is created, then the data will be available in a structured form. During the process the designer will group the data items, identify the primary keys, define the relationships (one-to-one, one-to-many or many-to-many), create the data model, normalize the data model and so on. Once the data model is created, it is verified against the proposed system in order to ascertain that the proposed model is capable of supporting the real-world system. So the data model is tested to find out whether the model can perform the various database operations (data loading, access, querying, insertion, modification and so on) and whether the data model takes care of the issues of data security, integrity and concurrency.

2.4.2 Logical Database Design

Once the different database management systems are evaluated and the one best suited for the organization is selected, the next step in the DDLC is the logical database design. Logical design is dependent on the choice of the database model that is used. Once the database model is identified, the conceptual design can be mapped to the logical design that is tailored to the selected database model. So the logical design is software dependent. In the logical design stage, the conceptual design is translated into internal model for the selected DBMS. This includes mapping all objects in the model to the specific constructs used by the selected database software. For example, for a RDBMS, the logical design includes the design of tables, indexes, views, transactions, access privileges, etc. Thus the logical design transforms the software-independent conceptual model into a software dependent model. The last section within this chapter previews the rules for translating a logical design of an ER diagram into a local relational schema.

2.4.3 Physical Database Design

Physical database design is the process of selecting the data storage and data access characteristics of the database. The storage characteristics depends on the type of devices supported by the hardware, the type of data access methods supported by the system.
and the DBMS. Physical design translates the logical design into hardware dependent one. Physical design is particularly important for older database models like hierarchical and network models. Relational, object-relational, object-oriented and deductive models are much more insulated from the physical layer details than the older database models. But even in the case of modern database models, physical design has great significance as a bad design can result in poor performance. In the case of distributed databases the physical design becomes more complex as the networking and communications issues also come into the picture.

2.5 Entity-Relationship Diagram

An Entity Relationship Diagram (ERD) is a visual language used to provide a graphical representation for the logical structure of a database. ER diagrams are easy to understand and do not require a person to undergo extensive training. Designers can use ER diagrams to easily communicate their ideas and insights with developers, customers, and end users. ER diagrams are readily translatable into relational tables which can be used to quickly build databases.

ER diagrams are initially proposed by Peter Chen in 1976 to create a uniform convention that considers both relational database and network views. Chen envisioned the ER model as a conceptual modeling approach that views real world data as systems of entities and relationships. Entities are data objects that maintain different relationships with each other. Additionally, entities are also described further using attributes. Since 1976, the ER model has been expanded and is sometimes used in business management, product development, and strategy formulations. However, database design remains its primary application.

There are three basic elements in an ER Diagram: entity, attribute, relationship. There are more elements which are based on the main elements as shown in Figure (2.2) including weak entity, multivalued attribute, derived attribute, weak relationship and recursive relationship. Cardinality and ordinality are two other notations used in ER diagrams to further define relationships.

2.5.1 Entity, Attribute & Relationship

The Entity can be a person, place, event, or object that is relevant to a given system. For example, a school system may include students, teachers, major courses, subjects, fees, and other items. Entities are represented in ER diagrams by a rectangle and named using singular nouns. A weak entity is an entity that depends on the existence of another entity. In more technical terms it is defined as an entity that cannot be identified by its own attributes. It uses a foreign key combined with its attributed to form the primary key. An entity like child is a good example for this. The child will be meaningless without
staff within the company so it depends on the existence of the staff entity. Weak entity is drawn inside a rectangle with a double line as shown in Figure (2.3).

An attribute is a property, trait, or characteristic of an entity, relationship, or another attribute. For example, the attribute Name is an attribute of the entity Student. An entity can have as many attributes as necessary. Note that some top level ER diagrams do not show attributes for the sake of simplicity. In those that do, however, attributes are represented by oval shapes. There are different types of attributes such as:

→ For attribute(s) that are used to distinguish an entity are called key attributes which uniquely identifies each entity. We underline key attributes with the Entity-Relationship diagrams. In the example shown in Figure(2.4), the id is the key attribute for the Staff.

→ If an attribute can have more than one value it is called an multivalued attribute. As the Staff entity can have multiple phone values or numbers. Multivalued attributes are drawn inside a double lined oval.
Derived attribute is an attribute based on another attribute. This is found rarely in ER diagrams. For example for the Age of the staff, it can be derived from the date of birth attribute. Derived attributes are drawn inside an oval with a dashed line.

Attributes can also have their own specific attributes. For example, the attribute address of the staff can have the attributes number, city, and state. These are called composite attributes which are illustrated in Figure (2.4).

![Attributes of E-R Diagram](image)

Figure 2.4: Attributes of E-R Diagram

A relationship describes how entities interact. For example, the entity Staff may be related to the entity Company by the relationship works. Relationships are represented by diamond shapes and are labeled using verbs as shown in Figure (2.5). Relationships can also have their own attributes to add further data to the relationship between the different entities. For example, in the Figure (2.6) shows that the relationship has an attribute called since for storing the date when the staff has started working for this company. Attributes for a relationship are drawn inside an oval.

Relationships within E-R diagrams are usually binary connecting two entities. But they can be used to connect more than two entities as shown in Figure (2.7) where the relationship work relates three different entities: Staff, Company and Location. If the same entity participates more than once in a relationship it is known as a recursive relationship. For instance, a staff can be a supervisor and be supervised, so there is a recursive relationship.

Cardinality specifies how many instances of an entity relate to one instance of another entity. Ordinality is also closely linked to cardinality. While cardinality specifies the occurrences of a relationship, ordinality describes the relationship as either mandatory or
optional. In other words, cardinality specifies the maximum number of relationships and ordinality specifies the absolute minimum number of relationships. When the minimum number is zero, the relationship is usually called optional and when the minimum number is one or more, the relationship is usually called mandatory. There are number of notations
used to present cardinality in ER diagrams. Chen, UML, Crow’s foot, Bachman are some of the popular notations.

To show the relationships between different entities lines in conjunction with the diamond shapes are used. It distinguishes between 1:1 (one to one), 1:M (one to many) and M:N (many to many) relationships using Chen notation. In addition there are numbers and letters to express the cardinality, which is the minimum and maximum number of related entities. A minimum cardinality of 0 at the same time also expresses that the relationship participation in this instance is optional. Figure (2.8) shows a comparison between Chen and Crow’s foot notations.

<table>
<thead>
<tr>
<th>Chen Notation</th>
<th>Crow’s Foot</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1,1)</td>
<td></td>
<td>One</td>
</tr>
<tr>
<td>0 (0,1)</td>
<td></td>
<td>Zero or One</td>
</tr>
<tr>
<td>M</td>
<td></td>
<td>Zero or More</td>
</tr>
<tr>
<td>M (1,M)</td>
<td></td>
<td>1 or More</td>
</tr>
</tbody>
</table>

Figure 2.8: Cardinality Notation for E-R Diagram

Figure (2.9) shows an example for an E-R diagram for a student entity which can attends only one course. Meanwhile, the course can be attended by a minimum of 10 and maximum of 20 students. The cardinality is expressed using the Chen notation.

The Crow’s Foot notation on the other hand according to Steward (2008) was developed by Clive Finkelstein. The most noticeable differences to the Chen notation is the use of a line only instead of the diamond to express the relationships and the inclusion of the attributes within the entity rectangle.

2.5.2 Enhanced ER Diagram

Contain all the basic modeling concepts of an ER Diagram adds additional concepts including: Specialization/generalization, Subclass/super class, Categories and Attribute inheritance. Extended ER diagrams use some object-oriented concepts such as inheritance. EER is used to model concepts more accurately than the ER diagram.
2.6 Relational Schema Design

The ER Model is intended as a description of real-world entities. Although it is constructed in such a way as to allow easy translation to the relational schema model, this is not an entirely trivial process. The ER diagram represents the conceptual level of database design meanwhile the relational schema is the logical level for the database design. For the sake of simplicity, we will be producing the relational schema for the following ER diagram:
2.6.1 Entities & Attributes

An entity type within ER diagram is turned into a table. You may preferably keep the same name for the entity or give it a sensible name but avoid DBMS reserved words as well as avoid the use of special characters. Each attribute turns into a column in the table. The key attribute of the entity is the primary key of the table which is usually underlined. It can be composite if required but can never be null. It is highly recommended that every table should start with its primary key attribute conventionally named in the format TablenameID. The initial relational schema is expressed in the following format writing the table names with the attributes list inside a parentheses as shown below for the corresponding ER diagram:

```
Company( CompanyID , name , address )
Staff( StaffID , dob , address )
Child( ChildID , name )
Wife ( WifeID , name )
Phone(PhoneID , phoneNumber )
Task ( TaskID , description)
```

Note that derived attributes are usually not included within the relational schema at this stage. Furthermore, within a relational database there is no such thing as a multi-valued attribute such as the phone within the ER diagram as attribute in logical relational schema, should be atomic. If you have a multi-valued attribute, take the attribute and turn it into a new entity or table of its own. Then make a 1:M relationship between the new entity and the existing one. Then convert as normal as explained in the next section.

2.6.2 Modeling Relationships

Expressing relationships at the logical level is a trivial process and not hard at all. At the end, a relationship can be expressed either using attributes or tables. There can be the following options depending on the cardinalities of relationships:

- 1:1 relationship: To keep it simple and even for better performances at data retrieval, I would personally recommend using attributes to represent such relationship. For instance, let us consider the case where the staff has or optionally has one wife. You can place the primary key of the wife within the table of the `Staff` which we call in this case Foreign key as shown below.

```
Staff( StaffID , dob , address , WifeID)
Wife ( WifeID , name )
```
Or vice versa to put the StaffID as a foreign key within the Wife table as shown below:

```
Staff( StaffID, dob, address )
Wife( WifeID, name, StaffID )
```

For cases when the staff is not married i.e. has no wifeID, the attribute can set to NULL.

- **1:M relationship**: This is the tricky part! For simplicity, use attributes in the same way as 1:1 relationship but we have only one choice as opposed to two choices. For instance, the staff can have from zero to many children, but a child can have only one staff as parent. To represent such relationship the StaffID as the Parent node must be placed within the Child table as a foreign key but not the other way around as shown next:

```
Staff( StaffID, dob, address, WifeID)
Child( ChildID, name, StaffID )
```

- **N:M Relationship**: We normally use tables to express such type of relationship. This is the same for N - ary relationship of ER diagrams. For instance, The staff performs many tasks whilst the task is performed by many staff. To express this relationship within a relational schema we use a separate table as shown below:

```
Staff( StaffID, dob, address, WifeID)
Task( TaskID, description)
Perform(PerformID, StaffID, TaskID )
```

- **Relationship with attributes**: It is recommended to use table to represent them to keep the design tidy and clean regardless of the cardinality of the relationship. For the instance for the relationship Work which has an attribute named Since, we will create a separate table including foreign keys for all related entities as shown below:

```
Staff( StaffID, dob, address, WifeID)
Company( CompanyID, name, address)
Work(WorkID, CompanyID, StaffID, since )
```
That’s the simple rules without any complexity! so the full relational schema is given below for the ER diagram:

```
Company (CompanyID, name, address)
Staff (StaffID, dob, address, WifeID)
Child (ChildID, name, StaffID)
Wife (WifeID, name)
Phone (PhoneID, phoneNumber, StaffID)
Task (TaskID, description)
Work (WorkID, CompanyID, StaffID, since)
Perform (PerformID, StaffID, TaskID)
```

### 2.7 Software for Database Design

There are a number of software tools available for database designers for generating the conceptual design of the database including:

- A free tool available on both Windows and Linux platform is *Dia* which can be downloaded from the internet at the site:

  ![Dia Website](http://www.dia-installer.de)

Dia is a general purpose drawing software for Windows, Mac OS X and Linux. Dia supports more than 30 different diagram types like flowcharts, network diagrams, database models. More than a thousand readymade objects help to draw professional diagrams. Software developers and database specialists can use Dia as a CASE tool to generate code skeletons from their drawings.

- *Creately.com* : is website dedicated for producing a variety of different diagrams including E-R and UML diagrams. Visitors can straightaway generate diagrams totally free of charge without the need to install any software. The website can be accessed at:

  ![Creately Website](http://www.creately.com)
2.8 Questions & Exercises

Try to have a go on the following questions:

1. What’s are the critical factors for choose a DBMS?
2. Why Requirement Analysis is the most important phase?
3. Give the resulting output for the three different design phases
4. What’s the difference between UML and ERD

Bibliography


Chapter 3

SQL : Structured Query Language

3.1 Introduction to SQL

Structured Query Language (SQL) is a programming language used for storing and managing data in relational database management systems. SQL was the first commercial language introduced for E.F Codd’s Relational model. Today almost all RDBMS such as MySQL, Oracle, Informix, Sybase and MS Access uses SQL as the standard database language. SQL is used to perform all type of data operations in RDBMS including table creation and data insertion. Although SQL is an ANSI (American National Standards Institute) standard, there are different versions of the SQL language depending on the type of the DBMS.

The SQL commands or statements can be arguably grouped into five major categories which are listed below. The majority of literature and books classify SQL statements into merely the first two categories mentioned here:

- **Data Definition Language (DDL)** which is mainly used for creating and defining databases and tables.
- **Data Manipulation Language (DML)** which is used for inserting, updating and deleting data from the database tables.
- **Data Query Language (DQL)** which is utilised to search and retrieve data from the database.
- **Data Control Language (DCL)** provides instructions to grant and take back authority i.e. access rights for or from the users for the database.
- **Transaction Control Language (TCL)** These commands are to keep a check on other commands and their effect on the database. These commands can cancel changes made by other commands by rolling back to the original state. It can also make changes permanent.
3.2 Data Definition Language

The main use of Data definition language is for creating, altering as well as deleting databases or tables. The main commands for the DDL are listed in the following table:

<table>
<thead>
<tr>
<th>SQL Command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>create schema</td>
<td>Create a database schema</td>
</tr>
<tr>
<td>drop schema</td>
<td>Drop a database schema</td>
</tr>
<tr>
<td>create table</td>
<td>Create a database table</td>
</tr>
<tr>
<td>drop table</td>
<td>Drop a database table</td>
</tr>
<tr>
<td>alter table</td>
<td>To modify the structure of a table</td>
</tr>
<tr>
<td>rename table</td>
<td>To change the name of a table</td>
</tr>
<tr>
<td>describe</td>
<td>To show the structure of a table</td>
</tr>
</tbody>
</table>

Table 3.1: SQL Data Definition Language.

3.2.1 CREATE

To create a database named *abc* we use the following command:

```
CREATE SCHEMA abc;
```

With MySQL, we can use the following statement:

```
CREATE DATABASE abc;
```

To view the list of names of existing databases, we use:

```
SHOW DATABASES;
```

In order to use the database named *abc*, the SQL statement is:

```
USE abc;
```

In order to create a database table named *hello*, we use the following command:
The created table `hello` has three attributes named: `id`, `name`, `description`. Table attributes can be named anything you want using alphabetical letters and numerical digits but avoid using special characters, accented letters or foreign letters. More importantly to make life easier for yourself, avoid the use of special SQL keywords such as `table`, `desc`, `like`, `select`...etc.

The keywords `INT`, `TEXT`, `VARCHAR` are basic database data types for the attributes. The `VARCHAR` means a set of characters where the number passed as argument 200, is the maximum numbers of characters. The different data types that are mostly used for creating attributes within most of the DBMSs include:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>INT</code></td>
<td>or <code>INTEGER</code> is a 32-bit signed integer value between: -2,147,483,648 and 2,147,483,647</td>
</tr>
<tr>
<td><code>BOOLEAN</code></td>
<td>stores TRUE or FALSE values</td>
</tr>
<tr>
<td><code>BIGINT</code></td>
<td>a 64-bit signed integer value with a very large range.</td>
</tr>
<tr>
<td><code>VARCHAR(N)</code></td>
<td>or <code>VARCHAR2(N)</code> is a string of variable length with a maximum of N characters.</td>
</tr>
<tr>
<td><code>CHAR(N)</code></td>
<td>a string of fixed length N.</td>
</tr>
<tr>
<td><code>TEXT</code></td>
<td>variable-length data.</td>
</tr>
<tr>
<td><code>FLOAT(size,d)</code></td>
<td>or <code>DOUBLE</code> is a decimal number such that:</td>
</tr>
<tr>
<td></td>
<td><code>size</code>: the maximum number of digits.</td>
</tr>
<tr>
<td></td>
<td><code>d</code>: max number of digits after the dot.</td>
</tr>
<tr>
<td><code>DATE</code></td>
<td>stores year, month, and day values.</td>
</tr>
<tr>
<td><code>TIME</code></td>
<td>stores the hour, minute, and second values.</td>
</tr>
<tr>
<td><code>TIMESTAMP</code></td>
<td>stores year, month, day, hour, minute, and second values</td>
</tr>
<tr>
<td><code>BLOB</code></td>
<td>a variable sized binary object with no size constraint.</td>
</tr>
<tr>
<td><code>ENUM(A,B,C,...)</code></td>
<td>restrict the attribute value to one of the given values: A, B or C ...</td>
</tr>
</tbody>
</table>

Table 3.2: SQL Data Types.
this attribute key. Another example to create a table named contacts within MySQL is shown next. The table has one primary key called contacts_id. Note that the primary key can be placed within the attribute declaration line. This is only valid for the case where the table has only one attribute as the primary key. We have different types of keys being outlined in this example.

```
CREATE TABLE IF NOT EXISTS contacts (
    contacts_id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name varchar(45) NOT NULL ,
    lastname varchar(45) NOT NULL,
    email varchar(100) ,
    age int DEFAULT '18',
    phone varchar(45) NOT NULL,
    UNIQUE KEY (email),
    KEY(phone),
    CONSTRAINT chk_age CHECK (age>17 AND age<61)
)
```

- **UNIQUE KEY** is for setting a Candidate Key that can be any column or a combination of columns that can qualify as unique key in the database. There can be multiple Candidate Keys in one table. Each but only one Candidate Key can qualify as Primary Key. You would get an error in case of an insertion of a new row with a duplicate or similar values for the candidate keys of an existing row in the database table.

- **PRIMARY KEY** is the combination of one or more column values in a table that make a row of data unique within the table. Primary keys are typically used to join related tables. Even if a table has no child table, a primary key can be used to disallow the entry of duplicate records into a table.

- **KEY** is a synonym for INDEX. Database index is a data structure that improves the speed of operations in a table. Indices can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.

The clause auto_increment is specific to MySQL only where an automated value is generated for the primary key when inserting data. For Microsoft Access, we use the word AUTOINCREMENT instead. In MSQL, we normally use IDENTITY keyword as shown below, which starts from the number 5 and increments each time by 2.
Oracle, does not have the keyword auto-increment, instead incrementing mechanism can done using sequences.

The **DEFAULT** constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified when inserting the data row into the database table. For instance, the value for the *age* column will be set by default to 18 when the age is not implicitly specified during the insertion of data.

The **CHECK** constraint is used to limit the value range or add a conditional constraint which can be placed in a column. For instance, we can create a new check constraint called *chk_person* to verify that data inserted should meet the conditions: the *lastname* value should be equal to *Nixon* whilst the *age* should be enclosed between 18 and 60 as shown in the statement below:

```sql
CONSTRAINT chk_person
    CHECK (lastname='Nixon' AND age between 18 AND 60)
```

### 3.2.2  **ALTER**

The SQL ALTER TABLE command is used to add, delete or modify columns in an existing table. You can also use ALTER TABLE command to add and drop various constraints on a an existing table. For instance to add an attribute named *email* of data type *VARCHAR* to the table named *clients*:

```sql
ALTER TABLE clients ADD email varchar(20)
```

To remove an attribute or a column named *age* from the table named *clients*:

```sql
ALTER TABLE clients DROP COLUMN age
```

To change the name of the attribute named *age* to *dob* with a new data type *date* within the table *clients*:

```sql
ALTER TABLE clients CHANGE age dob date NOT NULL
```

To add a constraint of check named *chk_age* for the table *contacts*, we use the following SQL statement:

```sql
"contacts_id int NOT NULL IDENTITY(5,2) PRIMARY KEY,
```
ALTER TABLE contacts
  ADD CONSTRAINT chk_age CHECK (age>17 AND age<71)

To drop a CHECK constraint which was added as above, use the following SQL statement under Oracle or MSQl:

ALTER TABLE contacts
  DROP CONSTRAINT chk_age

Under MySQL, we use the following statement for dropping a check constraint:

ALTER TABLE contacts DROP CHECK chk_age

### 3.2.3 DROP

The SQL DROP DATABASE statement is used to drop an existing database in SQL:

```
DROP SCHEMA abc;
```

The SQL DROP TABLE statement is used to remove:

- Table definition
- All data
- Indices
- Triggers
- Constraints
- Permission specifications.

The example below remove the table `clients` from the database:

```
DROP TABLE clients;
```
### 3.3 Data Manipulation Language

The data manipulation language (DML) is used for inserting, updating and deleting data from database tables. The SQL statements used for the DML are summarized within the following table. DML commands are used purely to modify data.

#### 3.3.1 INSERT

To insert a new data row or a set of rows into a database table:

```sql
INSERT INTO abc
  (column1, column2, , , columnN)
VALUES
  ('value1', 'value2', , , 'valueN');
```

Basically, it is optional to specify the list of attributes or column names but it is highly recommended or a must in some cases to write them down. To insert a row into the table named `clients` with the SQL shown below. Note that we do not need to specify the attribute `id` because if it is set `auto_increment`, it would be assigned automatically a value.

```sql
insert into clients ( name, email, city )
values
  ('Mike', 'mike@gmail.com', 'LA');
```

To insert a set of rows into the table `clients`. Note that also the use of the backslash `\` for Escaping special characters.
insert into clients ( name, email, city )
values
('Mike', 'mike@gmail.com', 'LA'),
('Adam', 'Adam@gmail.com', 'NY'),
('O\'nelle', 'mike@gmail.com', 'MN');

3.3.2 UPDATE

The SQL UPDATE query is used to modify existing records in a table. You can use the WHERE clause with UPDATE query to update selected rows otherwise all the rows would be affected. The syntax for the update query is shown below:

```
UPDATE table_name SET
    column1 = 'value1',
    column2 = 'value2',
    ....,
    columnN = 'valueN'
WHERE [condition];
```

For instance, to update the value of the attribute named email to a new value, for the client whose id is 6, we use:

```
UPDATE clients SET
    email = 'newemail@gmail.com'
WHERE id = 6;
```

We can use the logical operators within the WHERE clause such as AND and OR.

```
UPDATE clients SET
    email = 'newemail@gmail'
WHERE name='Adam' AND email='adam@gmail.com'
```

3.3.3 DELETE

The SQL DELETE query is used to delete existing records in a table. You can use the WHERE clause with the DELETE query to delete selected rows otherwise all the rows would be removed from the table. The syntax for delete is shown as:
DELECT FROM table_name
    WHERE [condition];

For example, to delete records within table clients whose id > 6:

DELECT FROM clients
    WHERE id > 6;

3.4 Data Query Language

Data Query Language (DQL) is utilised to search and retrieve data from the database. The main SQL statement within the DQL is the SELECT command:

<table>
<thead>
<tr>
<th>SQL Command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>select</td>
<td>To search data from a table</td>
</tr>
<tr>
<td>describe</td>
<td>To show the structure of a table</td>
</tr>
</tbody>
</table>

Table 3.4: SQL Data Query Language.

3.4.1 SELECT

SQL SELECT statement is used to search and retrieve the data from a database table. These result tables are called result-sets. The SELECT syntax is shown as:

SELECT column_list FROM table-name
    [WHERE Clause]
    [GROUP BY clause]
    [HAVING clause]
    [ORDER BY clause];

For a simple example, to displays all results for the table named clients showing only some specific attributes (id, name):

SELECT id, name FROM clients;

To retrieve all set of attributes (id, name, age, email ...), we use the wildcard *:
The SQL WHERE clause is used to specify a condition while fetching the data from single table or joining with multiple tables. If the given condition is satisfied then it returns specific values from the table. The WHERE clause is not only used in the SELECT statement, but it is also used within UPDATE and DELETE statements. As an example for the use of SELECT with the WHERE clause, we show how to get all clients whose id is equal to 6.

```sql
SELECT * FROM clients WHERE id=6
```

To show the id and email for clients whose id is greater than 6

```sql
SELECT id, email FROM clients WHERE id>6
```

To show the id, name, email for clients whose id is not equal to 6.

```sql
SELECT id, name, email FROM clients WHERE id <> 6
```

The <> operator (not equal) is equivalent to != as shown below:

```sql
SELECT id, name, email FROM clients WHERE id != 6
```

The SQL AND and OR operators are used to combine multiple conditions to narrow data in an SQL statement:

```sql
select * from table where
  (condition ) and (condition) or (condition)
```

For instance to show clients whose id is greater than 6 and their names is equal to 'abc':

```sql
SELECT * FROM clients
  WHERE id>6 AND name='abc'
```

To show clients having the condition as above or instead their email is equal to 'b@c':

```sql
SELECT id, email FROM clients
  WHERE
    (id>6 AND name='abc') OR email='b@c'
```
LIKE & NOT LIKE

The SQL LIKE clause is used to compare a value to similar values using wildcard operators. There are two wildcards used with the LIKE operator which are:

- The percent sign %: The percent sign represents zero, one, or multiple characters.
- The underscore _ : The underscore represents a single number or character.

To show all clients whose name attribute starts with a.

```
SELECT * FROM clients WHERE name LIKE 'a%'
```

To show all clients whose name attribute does not end with a.

```
SELECT * FROM clients WHERE name NOT LIKE '%a'
```

To show all clients whose name attribute contains a.

```
SELECT * FROM clients WHERE name LIKE '%a%
```

To show all clients where the name attribute, its second letter is a.

```
SELECT * FROM clients WHERE name LIKE '_a'
```

To show all clients whose name attribute has three letters and its second letter is a.

```
SELECT * FROM clients WHERE name LIKE '_a_
```

To show all clients whose name attribute which has at least one character.

```
SELECT * FROM clients WHERE name LIKE '_%
```

Set Operators

IN is a set operator that allows you to specify whether a value is contained within a list of multiple values. The IN operator is placed within the WHERE clause. The list of values can have a type of int or string.
SELECT * FROM clients WHERE 
client_id IN (3, 4, 9)

which is the equivalent of:

SELECT * FROM clients WHERE 
client_id=3 OR client_id=4 OR client_id=9

To negate the membership of a value within a list, we use: NOT IN.

SELECT * FROM clients WHERE 
client_id NOT IN (3, 4, 9)

which is the equivalent of using logical operators:

SELECT * FROM clients WHERE 
client_id<>3 AND client_id<>4 AND client_id<>9

The BETWEEN operator is used with the WHERE clause to select values within a specified range. The values can be numbers, text, or dates. The simple syntax for Between is given as:

```
SELECT column1, column2, 
FROM table_name 
WHERE column1 BETWEEN value1 AND value2;
```

For instance, to get all customers whose age attribute value is between 18 and 60 we use:

SELECT * FROM clients WHERE 
age BETWEEN 18 AND 60

**Aliases**

SQL aliases are used to give a temporary name for a database table or a column within a table. We usually use the AS operator to create an alias. For instance to give a temporary name total for the sum of prices of products, we use the following SQL statement:
To set an alias for a table name instead, we can use the following SQL statement for a table named `clients`. The temporary name for the table is given as `b`.

```
SELECT b.name, b.age FROM clients AS B
```

**ORDER BY**

The SQL ORDER BY clause is used to sort the data in ascending or descending order. The ordering is based on one or more columns. Some database management systems sorts results in ascending order by default. The syntax for ORDER BY within SELECT is shown below:

```
SELECT column-list FROM table_name
WHERE condition
[ORDER BY column1, column2, ...] [ASC | DESC];
```

For instance, to show clients and sort the retrieved results by their `id` attribute in an ascending (ASC) order:

```
SELECT * FROM clients ORDER BY id ASC
```

To show the list of clients but sort them in descending (DESC) order:

```
SELECT * FROM clients ORDER BY id DESC
```

To show the list of clients sorting by their name then by email when names are similar.

```
SELECT * FROM clients WHERE 1=1
ORDER BY name, email DESC
```

To randomize the order of the retrieved results, you can use the RAND() function as shown below. Be warned that this function can slow the performance of the DBMS for the query execution.

```
SELECT * FROM clients WHERE 1=1
ORDER BY RAND()
```
LIMIT or TOP or ROWNUM

The SQL TOP clause is used to fetch the TOP N number or X percent of records from a table. The TOP clause is not supported by all database management systems For example:

- MySQL supports LIMIT clause to fetch limited number of records.
- Oracle uses ROWNUM as a special variable to fetch a limited number of records.

For instance to show the first 3 rows from the clients table:

```
SELECT TOP 3 * FROM clients;
```

This is the equivalent of above by using MySQL:

```
SELECT * FROM clients LIMIT 3;
```

For Oracle to show the first three rows, we use:

```
SELECT * FROM clients WHERE ROWNUM <= 3;
```

In MySQL, to show the initial 10 rows from clients starting from the 3rd row:

```
SELECT * FROM clients LIMIT 3,10;
```

DISTINCT

The SQL DISTINCT keyword is used within the SELECT statement to eliminate all the duplicate records fetching only unique records. The syntax for using DISTINCT within the SELECT is shown below:

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition]
```

For instance to get only distinct names for customers from a table named clients:

```
SELECT DISTINCT name
FROM clients
WHERE age>18
```
3.4.2 DESCRIBE

The describe SQL statement is used to show the structure of a table via listing all of the fields in a table and the data format of each field. Note that the statement DESC is a synonym for DESCRIBE. For instance to show the structure of a table named clients, we use:

```sql
DESCRIBE clients;
```

3.5 Data Control Language

Data Control Language (DCL) commands are used to enforce database security in a multiple user database environment. There are two types of DCL commands which are GRANT and REVOKE. Only database administrator’s or owner’s of the database object can provide or remove privileges on a database object.

<table>
<thead>
<tr>
<th>SQL Command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>create user</td>
<td>creates a database user.</td>
</tr>
<tr>
<td>grant</td>
<td>grant permission or access rights to a database user.</td>
</tr>
<tr>
<td>revoke</td>
<td>To remove or take back a permission from a user.</td>
</tr>
</tbody>
</table>

Table 3.5: SQL Data Control Language.

3.5.1 CREATE USER

CREATE USER statement is to create a new user in a database server. To create a database user with the username emily and a password : boston201, we use the following SQL statement:

```sql
CREATE USER emily IDENTIFIED BY boston201 ;
```

The created user is limited to connect by default from localhost only. To allow the user emily to connect from any host we use the % wildcard, which usually means any host.

```sql
CREATE USER emily@% IDENTIFIED BY boston201 ;
```

To remove or drop the user from the database system, you can use:

```sql
DROP USER emily;
```
3.5.2 GRANT

SQL GRANT is a command used to provide access or privileges on the database objects to the users. The basic syntax for GRANT is shown below:

```
GRANT privilege_name
    ON object_name
    TO {user_name | PUBLIC | role_name}
    [WITH GRANT OPTION];
```

For instance to offer a user named *Imed* the SELECT and INSERT access rights to tables named *clients* and *products*, we use:

```
GRANT SELECT, INSERT
    ON clients, products
    TO Imed
```

3.5.3 REVOKE

The REVOKE command removes user access rights or privileges to the database objects. The Syntax for the REVOKE command is given as:

```
REVOKE privilege_name
    ON object_name
    FROM {user_name | PUBLIC | role_name}
```

For Example:

```
REVOKE SELECT ON clients FROM Imed;
```

This command will REVOKE the SELECT privilege on the *clients* table from the database user called *Imed*. When you REVOKE SELECT privilege on a table from a user, the user will not be able to SELECT data from that table anymore.
### 3.6 Transaction Control Language

Transaction Control Language (TCL) has a number of commands to keep a check on other SQL statements and their effects on the database. These commands can cancel changes made by other statements through rolling back to the original state. It can also make changes permanent. Transactions for SQL are further explained in the 5th chapter.

<table>
<thead>
<tr>
<th>SQL Command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>start transaction</td>
<td>to start a transaction.</td>
</tr>
<tr>
<td>commit</td>
<td>To permanently save data.</td>
</tr>
<tr>
<td>rollback</td>
<td>To undo the changes.</td>
</tr>
<tr>
<td>savepoint</td>
<td>To save changes temporarily.</td>
</tr>
</tbody>
</table>

Table 3.6: SQL Transaction Control Language.

### 3.7 Practice SQL

To get started to practice SQL statements covered in this chapter, you may either use HSQLDB or instead install EasyPHP which are both discussed in this section.

#### 3.7.1 Installing HSQLDB

HSQLDB is a lightweight relational database management system with size of 1 megabyte that can be used without any installation or complex configuration even from a USB drive. You can download the Jar file for the HSQLDB from the link below:

http://www.learndb.com/down/hsqldb.jar

Once saved to your computer, double click on the downloaded file [hsqldb.jar] to run it (do not extract the jar file). You need to have Java Runtime Environment (JRE) installed.
Click [OK] directly on the next screen to confirm the database user details as shown in Figure (3.1). You will get redirected to the main screen for the database management system as shown in Figure (3.2). The top section of the screen is for writing SQL queries: the bottom section for showing the results. Because every database management system has its own syntax for SQL as well as data types, we will be using the MySQL SQL version instead of the HSQLDB syntax. You need to type in this SQL command into the top SQL section and click [Execute SQL].
This command will allow HSQLDB to use the syntax of MySQL so that we can use auto_increment, text and so on. You can now type SQL commands as outlined in this chapter. For instance, to create a table called contacts, clear first the SQL window, and type in the following SQL statement to create the table, Click [Execute SQL]:

```sql
CREATE TABLE IF NOT EXISTS contacts (
  contacts_id int auto_increment primary key,
  name varchar(45) NOT NULL ,
  lastname varchar(45) NOT NULL,
  email text ,
  age int,
  phone varchar(45) NOT NULL
)
```

### 3.7.2 Installing WAMP with phpMyAdmin

MySQL is an open-source database management system used mainly for relational databases. It is considered the second mostly used open-source DBMS following SQLite. MySQL is being deployed for many high-profile and large scale website including Facebook, youtube, twitter and Flickr.

*phpMyAdmin* is a nice web application that is mainly used for creating, manipulating as well as managing MySQL databases through an easy web interface. *phpMyAdmin* usually requires Apache as a web server along with PHP integration. *EasyPHP* is a simple WAMP package( Windows with Apache, MySQL and PHP) that handles the installation of Apache webserver, MySQL, PHP as well as phpMyAdmin all together on Windows with a few simple clicks and without the need to manually download each software separately and handle complex configurations. To get started to install *EasyPHP* and use *phpMyAdmin* to practice SQL, follow the steps below:

1. Download EasyPHP DEVServer from the address:

   ![http://www.easyphp.org](http://www.easyphp.org)

2. Save it to your Computer and double click to install EasyPHP.

3. Once completed the installation, Launch EasyPHP . You can always launch it from: Start → All Programs → EasyPHP DevServer → *EasyPHP DevServer* ...
4. We need to launch PHPMyAdmin. Look for the EasyPHP icon on the bottom taskbar next to the Windows Watch as shown in Figure (3.4).

5. Right Click on the icon for EasyPHP on the taskbar and click on Administration as shown in Figure (3.4):

6. A webpage would show up on your Firefox or Chrome or Explorer. Look for phpMyAdmin under MODULES, Click Open to open phpMyAdmin
7. To create a new database, Click on Databases then type in the name of the database, then click Create.

![Figure 3.5: Creating a Database using phpMyAdmin.]

8. Once the database “databasename” is created, click on it within the left side:

9. On the next screen click on SQL, type in your SQL code and press GO.

![Figure 3.6: SQL Window for phpMyAdmin.]

3.8 Questions & Exercises

3.8.1 Part I : SQL Questions

Try to answer the following questions related to this chapter:

1. What’s the difference between Candidate key and Primary key ?

2. How to define two attributes (name & email) as one primary key in a table.

3. How to set auto increment for primary key within MySQL, MSQl and Access.

4. What’s the difference between the following data types: \textit{CHAR(10)} and \textit{VARCHAR(10)}.

5. Which data type can be set as KEY/Index in MySQL : \textit{VARCHAR} or \textit{TEXT}.

6. Which SQL functions can and cannot be placed within the \textit{WHERE} clause.

7. What’s the difference between MySQL and phpMyAdmin.

8. Give an example of SQL to enforce a constraint.

9. How to find the number of attributes within a table?

10. What’s the difference between : $<$, $!=$, $=$, LIKE ?

3.8.2 Part II : SQL Queries

We suppose that we have the following table named \textit{contacts} created.

\begin{verbatim}
CREATE TABLE IF NOT EXISTS contacts (  
    contacts_id int auto_increment primary key,  
    name varchar(45) NOT NULL ,  
    lastname varchar(45) NOT NULL,  
    email text ,  
    age int,  
    phone varchar(45) NOT NULL  
)
\end{verbatim}

Execute the following SQL insert query to fill the table with some data for testing :
You can answer the following question at the online web address:

[http://www.learndb.com/quiz/1.html](http://www.learndb.com/quiz/1.html)

1. Find the number of people inside the table.
2. Find the number of people whose AGE is 45
3. Find the number of contacts whose name starts with 'B'
4. Find the number of contacts whose name contains 'L' or 'l'
5. Find the number of people whose name ends with 'y'
6. Find the number of people whose age is between 40 and 50
7. Find the number of people where the second letter of their name is 'i'
8. Find the maximum age ?
9. Find the number of people whose name is composed of 5 letters.
10. Find the number of people whose age is either 41 or 45
11. Find the email of the person whose name is 'Bill' and lastname is 'Gates'
12. Find the phone number of the yougest person ( Having minimum age )
Bibliography


Chapter 4

Advanced SQL Queries

4.1 Aggregate Functions

The most useful SQL aggregate functions which usually return a single value, calculated from a set of values for a given column, are summarized in the following table. Note that aggregate functions **cannot** be used within the WHERE clause. To list the maximum

<table>
<thead>
<tr>
<th>SQL Function</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG()</td>
<td>Returns the average value</td>
</tr>
<tr>
<td>COUNT()</td>
<td>Returns the number of rows</td>
</tr>
<tr>
<td>SUM()</td>
<td>Returns the sum</td>
</tr>
<tr>
<td>FIRST()</td>
<td>Returns the first value</td>
</tr>
<tr>
<td>LAST()</td>
<td>Returns the last value</td>
</tr>
<tr>
<td>MAX()</td>
<td>Returns the largest value</td>
</tr>
<tr>
<td>MIN()</td>
<td>Returns the smallest value</td>
</tr>
</tbody>
</table>

Table 4.1: SQL Aggregate Functions.

price from the table named `products`:

```sql
select max(price) from products
```

To show the number of rows within the table `clients`. The column for the returned results is named `cc` because of the use of `AS`:

```sql
select count(*) as cc from clients
```

To list the average for the values of attribute `price` within the table `products`:
select avg(price) from products

To show the total sum of the price values from the table orders for the condition where clients_id=6:

select sum(price) from orders where clients_id=6

The FIRST() function is used to select the first retrieved row. However, it is only supported in Microsoft Access. For other database management systems such as MySQL which uses Limit whilst Oracle uses the special variable ROWNUM.

select first(price) from orders where clients_id=6

4.2 Scalar Functions

Every database management system has almost its built-in set of functions whilst the usage and syntax of such functions differ from vendor to vendor. As opposed to aggregate functions, scalar functions operate against a single value and return a single value. The following is set of most useful functions.

4.2.1 String Functions

- UCASE( ) : converts a text to upper case letters.

SELECT * FROM clients WHERE
UCASE(name) LIKE 'MIKE';

- LCASE ( ) : converts a text to lower case letters

SELECT * FROM clients WHERE
LCASE(name) LIKE 'mike';

- LENGTH( ) or LEN( ) : return the number of characters within a text.

SELECT * FROM clients WHERE
LENGTH(name) =4;
• MD5( ) : a hashing function to produce the MD5 value for a given string. The function is almost used for one-way encryption of confidential data such as passwords. For instance, to insert a new record into a table named users:

```sql
INSERT INTO users (email, pass) VALUES
('imed@imed.ws' ,md5('mysecret41'))
```

The inserted password won't be readable to anyone even to the database administrator. To find the inserted record using a given password, we use the following SELECT query:

```sql
SELECT * FROM users WHERE
   email LIKE 'imed@imed.ws' AND pass=md5('mysecret41')
```

• CONCAT( str1,str2,... ) : returns a string that results from concatenating the set of strings passed as arguments. For instance, to get the list of names concatenated with their ages for people whose name concatenated to their last name is equal to 'MarkCarter':

```sql
SELECT CONCAT(name, ' ', age) FROM users WHERE
   CONCAT(name, lastname) LIKE 'MarkCarter'
```

• TRIM( str1,str2,... ) : Removes leading and trailing spaces from a given string:

```sql
SELECT * FROM users WHERE
   TRIM(email) LIKE 'mark@soton.ac.uk'
```

• REGEX( ) : This is MySQL specific function for making a regular expression matching. For instance, to find all names starting with an alphabetical letter:

```sql
SELECT * FROM users WHERE
   name REGEX '[a-zA-Z]+'
4.2.2 Numerical Functions

• ROUND( ) : rounds a decimal number to an integer.

```
SELECT * from products where round(price)=4;
```

• SQRT( ) : returns the positive square root of a number.

```
SELECT * FROM products WHERE sqrt(price)=4;
```

• ABS( ) : returns the absolute value of a given number.

```
SELECT * FROM clients WHERE ABS(balance)=4;
```

• RAND( ) : is used to produce random numbers between 0 and 1. You can use ORDER BY RAND() to randomize the sorting of the returned results.

```
SELECT * FROM clients WHERE balance between rand()*100 and rand()*100
```

4.2.3 Time Functions

• NOW( ) : it returns the current date and time in the format : ‘2013-12-28 22:10:23’.

```
SELECT * FROM orders WHERE order_time < NOW()
```

• DATE( ) : it extracts the date part of a date or datetime expression. For instance, if we evaluate the expression DATE('2013-12-28 22:10:23'), it would return the value '2013-12-28'.
• YEAR( ) : it extracts the year part of a date or datetime expression.

There are further time functions such as MONTH(), DAY(), HOUR(), MINUTE() ...etc which are used to extract the corresponding part of a datetime expression.

4.3 GROUP BY

The SQL GROUP BY clause is used with the SELECT statement to arrange identical data into groups by a common value of a given attribute. Note that aggregate functions are used most of the time with the clause GROUP BY. The syntax for using group by is given below as:

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

For instance, to list the ids for countries and the number of clients within every country, we need to group the data by countries_id and apply a normal aggregate function count() to compute the number of clients grouped into the same countries_id as:

```
SELECT countries_id, count(id)
FROM clients
GROUP BY countries_id
```

In the same way, we list the total price paid by every customer from the table named orders. But in this example, we need to group by the clients_id:

```
SELECT clients_id, sum(price) AS cc
FROM orders
GROUP BY clients_id
```
4.4 HAVING

The HAVING statement is added to SQL because of the limitation where we can not use the aggregate functions (such as count, max ...) within the WHERE clause. The syntax for HAVING is given as:

```
SELECT column_name, aggregate_funct(column_name)
FROM table_name
WHERE conditions
GROUP BY column_name
HAVING aggregate_funct(column_name) operator value
ORDER BY column_name
```

For instance, to find the customer IDs for clients who have over 16 orders as shown in the example below. We need to use an aggregate function to count the number of orders but not inside the WHERE section. Note that the initial count named as total can not be placed within the WHERE condition. Therefore, we need to use HAVING clause:

```
SELECT clients_id, count(*) AS total
from orders
group by clients_id
having count(*) > 16
```

Note that non-aggregate SQL functions can be placed within the WHERE clause in contrast to aggregate functions which must be placed within the HAVING clause as shown in the example below in which the function LCASE is used to convert the name attribute to lower case:

```
SELECT clients_id, count(*) AS total
FROM orders WHERE
    LCASE(name) like 'laptop'
group by clients_id
having count(*) > 16
```

4.5 Nested Queries

A Subquery, Inner query or Nested query is a query within another SQL query which is usually embedded within the WHERE clause. A subquery is used to return data that
will be used in the main query as a condition to further restrict the data to be retrieved. Subqueries can be used with the `select`, `update`, `insert` and `delete` statements along with the operators like `=`, `<`, `>`, `>=`, `<=`, `IN`, `BETWEEN` etc. Subqueries are frequently used with the `select` statement as shown in the following syntax:

```sql
SELECT column_name FROM table1
WHERE
   column_name OPERATOR
   (SELECT column_name FROM table3 WHERE condition)
```

Note that an `ORDER BY` cannot be used in a nested query, although the main query can use an `ORDER BY`. To show an example for using nested query, we will find the list of customers who are from `Albania` and `Algeria` as:

```sql
SELECT clients.name, clients.last_name
FROM clients
WHERE
   clients.countries_id in
   ( SELECT countries_id FROM countries
      WHERE
         country LIKE 'Albania' OR
         country LIKE 'Algeria'
   )
```

In the same way as `SELECT`, subqueries can be used also within `DELETE` statement as shown in the example below:

```sql
DELETE FROM clients
WHERE countries_id in
   ( SELECT countries_id FROM countries
      WHERE
         country LIKE 'Albania' OR
         country LIKE 'Algeria'
   )
```

4.6 Join

The `JOIN` clause is used in an SQL query to retrieve data from two or more tables in a database. The `JOIN` combines the fields from different tables by using values common to
each other as the joining condition for the tables. ANSI-standard SQL outlines five different types of JOIN which are: INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER and CROSS. This is in addition to the special case of a self-join for a table.

Let us consider the following two tables: clients which has two attributes client_id and name. The table orders has four attributes order_id, client_id, descrip, amount. The client_id within orders references the same attribute within the clients table i.e. common attribute. Note that for the table orders, the record with order_id = 104 for a client_id = 7 that there is no corresponding client having such id within the clients table.

<table>
<thead>
<tr>
<th>clients_id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mark</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
</tr>
<tr>
<td>3</td>
<td>Tom</td>
</tr>
<tr>
<td>4</td>
<td>Marty</td>
</tr>
<tr>
<td>5</td>
<td>Emily</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>order_id</th>
<th>clients_id</th>
<th>descrip</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>Laptop</td>
<td>200</td>
</tr>
<tr>
<td>101</td>
<td>2</td>
<td>TV</td>
<td>300</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>Fridge</td>
<td>400</td>
</tr>
<tr>
<td>103</td>
<td>3</td>
<td>DVD</td>
<td>10</td>
</tr>
<tr>
<td>104</td>
<td>7</td>
<td>Monitor</td>
<td>90</td>
</tr>
</tbody>
</table>

Table : clients  
Table : orders

### 4.6.1 Inner Join

The INNER JOIN retrieves all rows from both tables as long as there is strictly a match between the columns in both tables. The syntax for INNER Join is written below:

```sql
SELECT table1.column1, table2.column2...
FROM table1
    INNER JOIN table2
    ON table1.common_fieldA = table2.common_fieldB
WHERE [conditions]
```

The INNER Join is also called simple Join with a simple syntax as shown below:

```sql
SELECT table1.column1, table2.column2...
FROM table1, table2
WHERE table1.common_fieldA = table2.common_fieldB
[AND conditions]
```

For example to get the list of customer names as well as the amount paid from both tables clients, orders with the condition that the amount paid must be over 5. The SQL statement is written below as:
SELECT clients.clients_id, clients.name, orders.amount
FROM clients
INNER JOIN orders
ON clients.clients_id = orders.clients_id
WHERE orders.amount > 5

The results for INNER Join returns only rows when there is a match in both tables. For instance, the records for clients Marty and Emily do not have a corresponding client_id with the orders table, thus they won’t be selected using INNER Join. The same is true for the order record who have a client_id = 7.

<table>
<thead>
<tr>
<th>client_id</th>
<th>name</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mark</td>
<td>200</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>Tom</td>
<td>10</td>
</tr>
</tbody>
</table>

An equi-join is a specific type of INNER join where comparatorison operator is the equal operator = being used in the joining condition as being shown in the example above which is the equivalent to the simplified version:

SELECT clients.clients_id, clients.name, orders.amount
FROM clients, orders
WHERE clients.clients_id = orders.clients_id
AND orders.amount > 5

For the case where the joining columns of both tables have the same name within an equi-join, the SQL statement can be shortened by employing the USING construct as shown below:

SELECT clients.clients_id, clients.name, orders.amount
FROM clients
INNER JOIN orders USING(clients_id)
WHERE orders.amount > 5

The natural join is another type of equi-join where the join predicate implicitly compares all columns in both tables that have the same names. Most experts don’t recommend the use of natural join. The SQL syntax for natural join is given below as:

SELECT clients.clients_id, clients.name, orders.amount
FROM clients
INNER JOIN orders USING(clients_id)
WHERE orders.amount > 5
4.6.2 Left Join

The LEFT JOIN which is called also LEFT OUTER JOIN, returns all rows from the left table considered as the primary table, with the matching rows in the right table. For the case where is no corresponding match within the right table, the result is set to NULL. The syntax for LEFT JOIN is given below as:

```sql
SELECT table1.column1, table2.column2...
FROM table1
    LEFT JOIN table2
    ON table1.common_fieldA = table2.common_fieldB
    WHERE [conditions]
```

For instance, to list ALL clients joined on their left with their paid amount, the following SQL is issued:

```sql
SELECT clients.clients_id, clients.name, orders.amount
FROM clients
    LEFT JOIN orders
    ON clients.clients_id = orders.clients_id
    WHERE 1=1
```

In this case, all clients would be shown, however for the two clients Marty , Emily would have their amount set to NULL.

<table>
<thead>
<tr>
<th>client_id</th>
<th>name</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mark</td>
<td>200</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>Tom</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Marty</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td>Emily</td>
<td>NULL</td>
</tr>
</tbody>
</table>
4.6.3 Right Join

The RIGHT JOIN (or RIGHT OUTER JOIN) is the opposite of LEFT JOIN whereby it returns all rows from the right table which is considered the primary table, with the matching rows from the left table. The result is set to NULL in the left side when there is no match within the right table. The syntax for the RIGHT JOIN is given below as:

```
SELECT table1.column1, table2.column2...
FROM table1
    RIGHT JOIN table2
    ON table1.common_fieldA = table2.common_fieldB
WHERE [conditions]
```

For example, to list all amounts from the table `orders` joined on their right, the list of names for the clients, the following SQL is used:

```
SELECT clients.clients_id, clients.name, orders.amount
FROM clients
    RIGHT JOIN orders
    ON clients.clients_id = orders.clients_id
WHERE 1=1
```

In this case, all orders would be shown even orders who don’t have a corresponding client in which case the `client_id` would be set to NULL. However, clients who do not have a corresponding order will not be selected.

```
<table>
<thead>
<tr>
<th>client_id</th>
<th>name</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mark</td>
<td>200</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>Tom</td>
<td>10</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>90</td>
</tr>
</tbody>
</table>
```

4.6.4 Full Join

The FULL JOIN (called also FULL OUTER JOIN) returns all the rows from the left table as well as results from the right table. The FULL JOIN is the result of both LEFT and RIGHT joins in a way that it shows results when there is a match in one of the tables. The syntax for FULL JOIN is given below as:

```
SELECT table1.column1, table2.column2...
FROM table1
    FULL JOIN table2
    ON table1.common_fieldA = table2.common_fieldB
WHERE [conditions]
```
For instance to show all clients as well as all amounts paid, we use:

```sql
SELECT clients.clients_id, clients.name, orders.amount
FROM clients
  FULL JOIN orders
  ON clients.clients_id = orders.clients_id
WHERE 1=1
UNION ALL
SELECT clients.clients_id, clients.name, orders.amount
FROM clients
  LEFT JOIN orders
  ON clients.clients_id = orders.clients_id
WHERE 1=1
UNION ALL
SELECT clients.clients_id, clients.name, orders.amount
FROM clients
  RIGHT JOIN orders
  ON clients.clients_id = orders.clients_id
WHERE 1=1
```

The results for the FULL join is shown below with a lot of duplicates:
4.7 SQL Views

SQL view is nothing more than a simple SQL statement that is stored in the database with a given name. The view is used to create a virtual table in the form of predefined SQL SELECT query. The view can contain all rows of a table or multiple joined tables or select a limited number of rows from the table(s). Views are very useful for the following reasons:

- Restrict or limit access to the data such that a user can see or even update what they need and no more.
- Summarize data from various tables which can be useful to generating reports.

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables, or another even another view. The basic CREATE VIEW syntax is as follows:

```
CREATE VIEW view_name AS
    SELECT column1, column2.....
    FROM table_name
    WHERE [condition];
```

For instance, to create a view named `monthly_sale` which used for creating a report for the total sum of sale on every month:
CREATE VIEW monthly_sale AS
    SELECT month, sum(amount) as total
    FROM orders
    WHERE status='Processed'
    GROUP BY month;

To execute a view to show data, it is considered exactly as a table as shown in the following SELECT statement:

SELECT * FROM monthly_sale

Note that other SQL statements such as UPDATE, INSERT and DELETE can be applied on a view in the same way as a table. For instance to update the retrieved total for the month of March to a new value of 1000, we use

UPDATE monthly_sale
    SET total = 1000
    WHERE month='March';

To drop the view from the database, we use:

SELECT * FROM monthly_sale;

4.8 Union

The SQL UNION operator combines the result of two or more SELECT statements. Each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL

SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;

For the UNION ALL syntax is given as:
In the following example, we list all the cities for the clients unioned with the list of cities for the suppliers as given below:

```sql
SELECT city, country FROM clients
  WHERE country='Germany'
UNION ALL
SELECT city, country FROM suppliers
  WHERE country='Germany'
ORDER BY City;
```

### 4.9 Intersect

The INTERSECT statement returns only common rows returned by the two SELECT statements. The same rules for UNION apply to the INTERSECT operator. MySQL does not support the INTERSECT operator. The syntax for intersect is given below as:

```sql
SELECT column1 [, column2 ] FROM table1 [, table2 ]
[WHERE condition]
INTERSECT
SELECT column1 [, column2 ] FROM table1 [, table2 ]
[WHERE condition]
```

### 4.10 Foreign Keys

A foreign key is a key used to link two tables together. This is sometimes called a referencing key. Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table. The SQL FOREIGN KEY CONSTRAINT is used to ensure the referential integrity of the data in one table to match values in another table. The values of the FOREIGN KEY columns in each row of the referencing table have to match with the values of the corresponding primary key columns of a row in the referenced table.
Let us consider the case where we have two tabled named `friends` and `phones` for a simple Phonebook database. The `phones` table has an attribute `friend_id` which references the primary key `friend_id` of the `friends` table as shown below:

<table>
<thead>
<tr>
<th>friend_id</th>
<th>name</th>
<th>phone_id</th>
<th>friend_id</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mark</td>
<td>100</td>
<td>1</td>
<td>023801111</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>101</td>
<td>2</td>
<td>043929292</td>
</tr>
<tr>
<td>3</td>
<td>Tom</td>
<td>102</td>
<td>3</td>
<td>022102022</td>
</tr>
<tr>
<td>4</td>
<td>Marty</td>
<td>103</td>
<td>4</td>
<td>012020390</td>
</tr>
<tr>
<td>5</td>
<td>Emily</td>
<td>104</td>
<td>5</td>
<td>039200022</td>
</tr>
</tbody>
</table>

Table: `friends`  
Table: `phones`

Suppose now, we want to delete the person `Mark` from the `friends` table.

```sql
DELETE FROM friends where name='Mark'
```

What would happen to his phone numbers? we have to manually delete them as shown below:

```sql
DELETE FROM phones where friend_id=1
```

But if the database design is complex and there are many other tables which are dependent on this record, we have to go through them and delete them manually otherwise the database would be in a inconsistent state. Further, there are situation where the is a policy not to delete records if there is a sensitive data that must be kept. For instance payment details or staff activity logs and so on.

The same scenario happens for updating data if we update the `id` of Emily for instance to a new value of 7.

```sql
UPDATE friends SET friend_id=7 where name='Emily'
```

In the same way, all phones or data pointing to Emily has to be updated in which case it becomes a difficult process for larger databases.

```sql
UPDATE phones SET friend_id=7 where friend_id=5
```

This is why we need to use Foreign keys to ease the management of referential integrity between different tables. To create tables and explicitly specify the foreign keys, we would use the following SQL statements in order to create the two tables: `friends` and `phones`. 

*Learn Databases - 1st Ed.*  
*www.LearnDB.com*  
*I. Bouchrika, 2014*
create table friends (  
  friend_id int not null auto_increment,  
  name varchar(100) not null ,  
  PRIMARY KEY(friend_id)  
);  

For the table *phones* which has the Foreign key:

```
create table phones (  
  phone_id int not null auto_increment primary key,  
  friend_id int not null,  
  phone varchar(100) not null,  
  PRIMARY KEY(phone_id),  
  FOREIGN KEY friend_id  
    REFERENCES friends(friends_id)  
    ON DELETE RESTRICT  
    ON UPDATE CASCADE  
)TYPE=InnoDB;
```

Instead, you can use the ALTER command which is implemented on most DBMS for adding a foreign key to a table:

```
ALTER TABLE phones  
  ADD FOREIGN KEY (friend_id) REFERENCES friends(friends_id)  
  ON DELETE RESTRICT  
  ON UPDATE CASCADE ;
```

Note that the database engine for MySQL must be *InnoDB*. To have an existing an existing table in MySQL its type changed to use *InnoDB* engine, you can use the ALTER command as shown below:

```
ALTER TABLE contacts TYPE=INNODB;
```

For the case of an SQL delete, it would give us an error for the following SQL statement:

```
DELETE FROM phones where friend_id=1
```
This is because of the use of keyword RESTRICT in which case we need to explicitly delete all dependent phone numbers first. For the case of update, if we update the \textit{friend}_\textit{id} of \textit{emily}, the database management system would automatically update all dependent data. This is because of the use of the keyword CASCADE. The list of keywords or options being used for the foreign keys specification are explained below:

- **ON DELETE CASCADE:**
  When a row in the parent table is deleted, InnoDB will automatically delete corresponding foreign key column in the child table.

- **ON DELETE SET NULL:**
  When a row in the parent table is deleted, InnoDB will automatically set corresponding foreign key column in the child table to NULL.

- **ON DELETE RESTRICT:**
  It disallows a delete if an associated record still exists.

- **ON UPDATE CASCADE:**
  Update corresponding foreign key column in all matching rows in the child table to the same value.

- **ON UPDATE SET NULL:**
  Sets corresponding foreign key column in all matching rows in the child table to NULL.

- **ON UPDATE RESTRICT:**
  Disallows an update if an associated record still exists.

### 4.11 Questions & Exercises

#### 4.11.1 Part I: SQL Questions

Try to answer the following questions related to this chapter:

1. How to get only the first row using SELECT statement within MySQL?
2. How to search for a string value which contains only digits using MySQL?
3. How to pickup a single row at random in MySQL.
4. What’s the difference between scalar and aggregate functions?
5. Can we place a scalar function within the HAVING clause?
6. What’s the difference between Simple Join and Left Join?
7. How to perform an SQL Full Join in MySQL
8. What’s a view?
9. How to create view?
10. What’s the difference between CASCADE and RESTRICT

4.11.2 Part II: SQL Queries

You can answer the following question at the online web address:

http://www.learndb.com/quiz/2.html

Given the logical structure of the following four tables:

clients(clients_id, countries_id, name, last_name)
countries(countries_id, country)
products(products_id, name, price)
orders(orders_id, clients_id, products_id, price, o_time)

The database SQL file can be downloaded and imported to your dbms at:

http://www.learndb.com/down/ecommerce.sql

A brief sample for the database showing the different attribute values is below as:

INSERT INTO clients (clients_id, name, last_name, countries_id) VALUES (1, 'Mark', 'Nixon', 77), ...
INSERT INTO countries (countries_id, country) VALUES (1, 'Afghanistan'), ...
INSERT INTO products (products_id, name, price) VALUES (2, 'European Orange', 30), ...
INSERT INTO orders (orders_id, clients_id, products_id, price, o_time) VALUES (1, 1, 1, 10, '2012-11-07'), ...
Answer the following questions through the use of SQL:

1. Find the number of countries that ends with ia
2. Find the number of countries that has 6 letters and ends with ia
3. Find the number of countries which has the letter a or h
4. Find the number of clients whose names contain h and r
5. Find the number of clients from Algeria
6. Find the number of countries whose second letter is "l"
7. Find the number of distinct names for the clients
8. Find the number of clients who are from countries that end with a
9. Find the total sum of orders for clients who are from countries that ends with a
10. Find the number of countries
11. Find the number of clients who do not have an order
12. Find the number of countries which don’t have any order
13. Find the number of orders bought from clients of Canada in Junes
14. Using Group by, limit, order by and count, find the most ordered product
15. Using Group by, limit, order by and count, find the most ordered product in Austria
16. Find the number of orders in 02/2012
17. Which is the date having the largest revenue
18. Find the number of customers who have over 5 orders.
19. Find the number of clients who have paid over 1000 in total
20. Find the difference between the total sale of 2011 and 2010
Bibliography


Chapter 5

SQL Transactions

5.1 Introduction

Transaction is defined as a collection of actions or statements that transforms the database from one consistent state into another consistent state. The transaction contains a number of SQL statements. After the transaction begins, all of the SQL statements within the transaction are executed sequentially. At the end of the transaction, either permanent changes are made into the database tables for all executed SQL statements or instead the database is rolled back to its initial state. Transactions have the following four standard properties, usually referred to by the acronym ACID:

- **Atomicity**: ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their initial state.

- **Consistency**: ensures that any transaction will bring the database from one valid state to another valid state. Any data written to the database must be valid according to all defined rules, including but not limited to constraints, cascades, triggers and so on.

- **Isolation**: ensures that results obtained after the concurrent and parallel execution of transactions, would be the same results obtained if transactions were executed serially, i.e. one after the other.

- **Durability**: ensures that the result or effect of a committed transaction persists i.e. stored permanently into the database in case of a system failure.

Let’s consider the simplest case for transactions. We suppose there is a bank having only two clients Mark and John with balances of $100 and $0 respectively. The total deposit by clients within the bank is $100. Mark wants to transfer an amount of $100
from his bank account to the account of John. After the transfer, the total deposit within the bank should stay as it is: $100. In database language, it would be done normally using two SQL statements. The first statement for deducting the amount from Mark’s account as shown below:

```
UPDATE account SET balance=balance-100
WHERE name='Mark';
```

The other SQL statement would be of course to add the amount to John’s account as:

```
UPDATE account SET balance=balance+100
WHERE name='John'
```

But suppose that just after the first SQL statement has been executed and saved successfully, the server crashes or there was a network error, what would happen? An amount of $100 is deducted from the account of Mark, meanwhile, John’s account is kept as it is. Thus, an amount of $100 is lost when the server went offline. The database state is not consistent in this case as the total balance of accounts would not be the same as the amount of available cash i.e. $100. Ideally, all SQL statements should executed and their effects stored permanently or none of the transactions is executed at all to keep the database at its consistent state. This is the atomicity aspect of transactions: Either all or none.

### 5.2 Isolation Levels

The isolation of transactions is ideally to ensure that the results obtained after the concurrent and parallel execution of transactions, would be the same results obtained if transactions were executed serially, i.e. one after the other. There are different levels of isolations being offered by the DBMS to ensure that transactions operate independently and transparently. Before we explain the different levels of isolation, we need to shed light on the main issues called Three R’s that occur in a highly concurrent environment:

1. **Dirty Read**: One transaction (I) reads changes made by another transaction (II) which has not yet been committed. However, if the second transaction (II) rolls back, then data which the first transaction (I) has read is not correct. Let’s consider a bank account table with an initial balance of $1000 and we want to add interest to the account with a rate of 20% using Transaction I. There is another Transaction II running in parallel which attempts to update the balance to $100 but later on rollbacks its effects as shown in the following table. The order of SQL statements executed within both interleaved transactions is illustrated in Table (5.1). We note that interest is added based on the wrong balance of $100 instead of $1000 because of the dirty read.
2. **Unrepeatable read**: it usually occurs due to table UPDATEs committed by other transactions. A transaction reads a row twice and reads different state each time because another transaction may have written to the row, and committed, between the two reads. For example, a transaction (I) reads initially the balance from the account table as $1000. The other transaction (II) updates the values of the balance to $100 and commit its changes to the database as shown in Table (5.2) for the two interleaved transactions. If transaction (1) makes another select to read the balance value, it would get a different value of $1000 as shown in step 5. So which one to use to compute the interest rate:

<table>
<thead>
<tr>
<th>Transaction I</th>
<th>Transaction II</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: SELECT balance FROM account</td>
<td>1: UPDATE account SET Balance=100</td>
</tr>
<tr>
<td>Balance value is now $1000</td>
<td></td>
</tr>
<tr>
<td>2: SELECT balance FROM account</td>
<td>2: UPDATE account SET Balance=100</td>
</tr>
<tr>
<td>myvar1=balance*1.20</td>
<td>4: COMMIT</td>
</tr>
<tr>
<td>Balance is back to $1000</td>
<td></td>
</tr>
<tr>
<td>5:UPDATE account SET balance=myvar1</td>
<td>5: UPDATE account SET balance=myvar1</td>
</tr>
<tr>
<td>Interest is added based on $100</td>
<td>6: UPDATE account SET balance=myvar1</td>
</tr>
<tr>
<td></td>
<td>Interest is added based on $1000</td>
</tr>
</tbody>
</table>

Table 5.1: Dirty Read For Transactions

3. **Phantom read**: This occurs due to new records inserted and committed by other transactions. A transaction executes a query twice, and the second result set includes rows that were not visible in the first result set. This situation is caused by another
transaction inserting new rows between the executions of the two queries.

<table>
<thead>
<tr>
<th>Transaction I</th>
<th>Transaction II</th>
</tr>
</thead>
</table>
| 1: SELECT sum(balance) FROM account  
  Total Balance is $1000  
  myvar1=balance*1.20 | 2: INSERT INTO account  
  (balance) VALUES (2000) |
| 4: COMMIT  
  Total Balance is $2000 | |
| 5: SELECT sum(balance) FROM account  
  Total Balance is $1000  
  myvar2=balance*1.20 | 6: UPDATE ....  
  based on : myvar1 or myvar2 ? |

Table 5.3: Phantom Read For Transactions

In database systems, most DBMS provide different level of isolation to handle the issues explained earlier. In a highly concurrent environment, highly isolated transactions can lead to deadlocks. A deadlock is a situation, where transactions compete over shared resources and effectively prevent each other from accessing the resource whilst waiting for other to release the resource. Furthermore there is a tradeoff between isolation level and performance of the database system. Therefore, database systems provide several levels of isolation for transactions. For instance, MySQL offers four levels of transaction isolation: which are outlined as follows:

1. Serializable: In the serializable isolation level all transactions occur in a completely isolated fashion. All transactions are executed serially one after the other.

2. Repeatable read: In a repeatable read isolation level, the statements cannot read data that has been modified but not yet committed by other competing transactions. No other transactions can modify data that has been read by the current transaction until the current transaction completes. It is the default isolation level for most DBMS including MySQL.

3. Read committed: In a read committed isolation level, SQL statements cannot read data that has been modified but not committed by other transactions. Statements wait until rows of data that are write-locked by other transactions are unlocked before they obtain their own locks. This prevents them for having dirty read.

4. Read uncommitted: In a read uncommitted isolation level, the statements can read rows that have been modified by other transactions but not yet committed.
A summary for the four different level of isolation is shown in the following tables with their possible handling for the three Rs issues.

<table>
<thead>
<tr>
<th></th>
<th>Dirty Read</th>
<th>Unrepeatable Read</th>
<th>Phantom Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serializable</td>
<td>Not Possible</td>
<td>Not Possible</td>
<td>Not Possible</td>
</tr>
<tr>
<td>Repeatable read</td>
<td>Not Possible</td>
<td>Not Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Read committed</td>
<td>Not Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Read uncommitted</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
</tbody>
</table>

Table 5.4: Phantom Read For Transactions

5.3 Transaction Statements

In most database management systems, to start a transaction, we use the command:

```
START TRANSACTION;
```

To commit, the changes made by a transaction, we use: or to set the level

```
COMMIT;
```

To undo or cancel the effects of SQL statements within a transaction, we call:

```
ROLLBACK;
```

For instance, for the initial example of the fund transfer between two accounts, we use a transaction as illustrated below:

```
START TRANSACTION;
UPDATE account SET balance=balance-100 WHERE name='Mark';
UPDATE account SET balance=balance+100 WHERE name='John';
COMMIT;
```

Note that in case you are using MySQL, you must use the InnoDB database engine when creating table as it does support transactions. For example, to create a table called account within MySQL, we use the following statement:

```sql
CREATE TABLE account (...
```
CREATE TABLE account(
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY
    name VARCHAR(100) NOT NULL,
    balance INT NOT NULL DEFAULT 0
) TYPE=InnoDB;

To have an existing table in MySQL its type changed to use InnoDB engine, you can use the ALTER command as shown below:

ALTER TABLE contacts TYPE=INNODB;

Further, in MySQL, you need to set the auto-commit to the value 0 so that SQL statements within a transaction would be saved permanently only when the commit is submitted explicity. The disable the auto-commit use the following statement:

SET AUTOCOMMIT=0

To check or retrieve the value of AUTO COMMIT from the DBMS, we use the command:

SELECT @@autocommit;

To undo the effects of SQL statments for a transaction that has not been committed yet, we use the command ROLLBACK as shown below:

START TRANSACTION;
    UPDATE account SET balance=balance-100 WHERE name='Mark';
    UPDATE account SET balance=balance+100 WHERE name='John';
ROLLBACK;

MySQL makes it possible to control the isolation level via the special TRANSACTION ISOLATION LEVEL variable. You can obtain the current value of this variable at any time with a fast SELECT, as below:

SELECT @@tx_isolation;

To set a different level level of isolation, you can use the following command to change it to the level of SERIALIZABLE:
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Other level values include: READ UNCOMMITTED, READ COMMITTED and REPEATABLE READ.

There are further topics for handling and managing transactions such as locking tables as well as rows or using savepoint which are beyond the scope of this chapter for the sake of keeping it simple.

5.4 Exercises & Questions

Try to answer the following questions

1. What’s a database transaction?
2. What are the four properties of a transaction?
3. Give an example for the atomicity property.
4. How to start and commit a transaction?
5. How to cancel the effects of SQL statements within a transaction?
6. What are the three Rs?
7. What’s a deadlock?
8. What are the different level of isolation provided by the DBMS?
9. What’s the default isolation level for MySQL?
10. How to stop transaction to interleave each other?

Bibliography


Chapter 6

Procedures & Triggers

6.1 Procedures

A stored procedure is a set of SQL statements stored inside the database. The stored procedure can be invoked and used by triggers, other stored procedures or applications such as Java, C#, PHP, etc. The main use for stored procedures is to help to reduce the traffic between application and database server. Instead of sending multiple SQL statements for processing, the application has to send only the name of the stored procedure with any parameters. The stored procedures are usually reusable and transparent to any applications. We will be going through MySQL procedure in this chapter whilst Oracle procedure are further explained in Chapter 6.

The SQL statement to create a procedure is CREATE PROCEDURE. For example, in order to create a procedure that returns the list of clients having an age over 18 is given below as:

```
DELIMITER //
CREATE PROCEDURE GetClientsList()
BEGIN
    SELECT * FROM clients
    WHERE age>18
END
//
```

The first command is DELIMITER //, which is not related to the stored procedure syntax. The DELIMITER statement changes the standard line-termination operator which is normally semicolon (;) to a new given one. In this case, the delimiter is changed from the semicolon(;) to double-slashes ///. This is done so that we can comfortably use the semi-colon as the delimiter within the procedure body so that it can be passed to the
server as a whole rather than letting the DBMS to be interrupted at each statement. After the END keyword, we use the new delimiter // to indicate the end of body for the stored procedure.

The CREATE PROCEDURE statement used to create a new stored procedure followed by the name of the procedure which is GetClientsList. We put the parentheses after the name of the stored procedure. The section enclosed between BEGIN and END is for the body of the stored procedure which contains the core implementation of the function. In other words, the procedure body contains the declarative SQL statements to handle business logic. In this stored procedure, we use a simple SELECT statement to query data from the clients table.

More importantly, You must afterwards change the delimiter back to the semi-colon (;) as shown below:

```
DELIMITER ;
```

To call the stored procedure, you can use the following command:

```
CALL PROCEDURE GetClientsList();
```

To view the list of existing procedures within MySQL, the SHOW statement is used as:

```
SHOW PROCEDURE STATUS;
```

To drop a procedure from the database, you can use:

```
DROP PROCEDURE IF EXISTS GetClientsList;
```

### 6.1.1 Procedure Variables

A variable is a named data object that references a certain value. The variable value can change during the execution of the stored procedure. We typically use the variables within stored procedures to hold information or data that assist with the calculation. These variables are local to the stored procedure and cannot be accessed elsewhere. To declare a variable inside a procedure, we use the DECLARE statement as follows:

```
DECLARE variable_name datatype(size) DEFAULT default_value
```
For example, we can declare a variable named `total_sale` with the data type INT and default value of 0 as follows:

```sql
DECLARE total_sale INT DEFAULT 0
```

To declare another variable named `country` as VARCHAR with a default value of `Finland`, we use:

```sql
DECLARE country VARCHAR(100) DEFAULT "Finland"
```

To assign a value to a variable, the SET keyword is employed as shown in the example below:

```sql
SET total_sale=10
```

Besides the SET statement, you can use SELECT INTO statement to assign the result of a query to a variable. Notice that the query must return a scalar value i.e. single value not a collection.

```sql
DECLARE total_sale INT DEFAULT 0
SELECT SUM(*) INTO total_sale FROM orders
```

A variable that begins with the `@` sign at the beginning is session variable. It is available and accessible until the session ends. In other programming languages such as PHP is called a Global variable.

### 6.1.2 Procedure Parameters

Most of the stored procedures that are developed require parameters as well as a return a value. The parameters passed into the stored procedure add further flexibility and control. In MySQL, a parameter has one of three modes IN, OUT or INOUT which are explained as follow:

1. **IN**: is the default parameter mode. For the case where a parameter is defined as IN parameter in a stored procedure, the program or user invoking the procedure has to pass an argument to the procedure. Furthermore, the value of an IN parameter is usually protected in a way that even if the value of the IN parameter is changed inside the stored procedure, its original value is kept after the stored procedure ends. In other words, when executing the procedure, only a copy of the IN parameter is taken which we call in programming "Pass by value".
The following stored procedure named `getClientByCounty` takes one IN parameter
named `country` with a data type of `VARCHAR`:

```sql
DELIMITER //
CREATE PROCEDURE getClientsByCountry(IN country VARCHAR(255))
BEGIN
    SELECT *
    FROM clients
    WHERE countryName = country;
END //
DELIMITER ;
```

To call the procedure named `getClientByCounty`, we use the following statement:

```sql
CALL PROCEDURE getClientsByCountry('Brazil');
```

2. **OUT**: The OUT parameter can be considered as a return variable for the stored
procedure. The value of an OUT parameter can be assigned and changed inside the
stored procedure whereafter the new value is passed back to the calling program. The
initial value of the OUT parameter must be assigned within the stored procedure.
For example, we define a new procedure named `countClientByCountry` that would
take two parameters. The OUT parameter is named `result` which stores the total
number of clients for a given country.

```sql
DELIMITER //
CREATE PROCEDURE countClientByCountry(
    IN country VARCHAR(25),
    OUT result INT)
BEGIN
    SELECT count(*)
    INTO result
    FROM clients
    WHERE countryName = country;
END //
DELIMITER ;
```

To call such function, we need initially to create a new variable called for instance
`@hello` using the SET command. Afterwards, to view the value assigned to the OUT
variable, we use fast select as shown below:
SET @hello=0;
CALL PROCEDURE countClientByCountry('Brazil', @hello);
SELECT @hello;

3. **INOUT** : INOUT parameter is the combination of both IN and OUT parameters. It means that the calling program can pass an argument, and the stored procedure can modify such parameter and pass the new value back to the calling program.

```sql
DELIMITER //
CREATE PROCEDURE set_counter(INOUT mycount INT(4))
BEGIN
    SET mycount = mycount + 1;
END //
DELIMITER ;
```

The `set_counter` stored procedure accepts one INOUT parameter `mycount`. To invoke the `set_counter` stored procedure with a global variable named `@counter` already initialized the value 2

```sql
SET @counter = 2;
CALL set_counter(@counter);
CALL set_counter(@counter);
CALL set_counter(@counter);
SELECT @counter - would give 5;
```

### 6.1.3 Flow Control

**IF**

The IF statement allows you to execute a set of SQL statements based on the evaluation of a certain condition or value of an expression. To form an expression in MySQL, you can use variables, operators, and even functions as long as the expression can return a boolean value of either TRUE or FALSE. The following illustrates the syntax of the IF statement for stored procedures. Note that the use of `[ ]` which means that it is optional.
IF if_expression THEN commands
    [ELSEIF elseif_expression THEN commands]
    [ELSE commands]
END IF;

For example, we create a stored procedure named `countClientByStatus` that returns the number of customers based on their `confirm` status. The procedure is passed as an IN parameter named `status` and returns the result via OUT variable. Note that the comparison operator is =, the same as the assignment operator.

```
DELIMITER //
CREATE PROCEDURE countClientByStatus(
    IN status INT,
    OUT result INT)
BEGIN
    IF status = 0 THEN
        SELECT count(*) INTO result FROM clients
        WHERE confirm = 'NO';
    ELSEIF status = 1 THEN
        SELECT count(*) INTO result FROM clients
        WHERE confirm = 'YES';
    ELSE
        SET result=0 ;
    END IF
END //
DELIMITER ;
```

**CASE**

Besides the IF statement, most DBMSs provide an alternative conditional statement called CASE. The CASE statement is better than IF as it makes the code more readable and efficient when there are a large number of cases. The syntax of the CASE statement is given below. The `case_expression` is usually a variable that can take one of the values given inside the CASE as `when_expression_1`. For the case where there are no matches, the ELSE command is executed.
The following example shows a procedure named `computeTaxRate` based on a given country returning the results via OUT variable.

```sql
DELIMITER //
CREATE PROCEDURE computeTaxRate(
    IN country VARCHAR(100),
    OUT result INT)
BEGIN
    CASE country
        WHEN 'USA' THEN
            SET result = 17;
        WHEN 'FRANCE' THEN
            SET result = 27;
        ELSE
            SET result = 20;
    END CASE;
END//
DELIMITER ;
```

### 6.1.4 Loops

Programming loop statements allow you to execute a block of code repeatedly or iteratively based on a condition. There are three loop statements within MySQL: WHILE, REPEAT and LOOP. In this chapter, we consider only the first two.

**WHILE**

The WHILE loop checks the `expression` predicate at the beginning of each iteration. If the `expression` evaluates to TRUE, the DBMS engine will execute the set of statements between WHILE and END WHILE until the `expression` is evaluated to FALSE. The WHILE loop is called `pretest` loop because it checks the expression before the statements execute. The syntax of the WHILE loop statement is given below as follows:
WHILE expression DO
  Statements
END WHILE

REPEAT

In contrast to the WHILE loop, the DBMS executes initially the statements, and then it evaluates the expression within the REPEAT loop. If the expression evaluates to TRUE, the DBMS executes the statements iteratively until the expression evaluates to FALSE. The syntax of the REPEAT loop statement is given as:

REPEAT
  Statements;
UNTIL expression
END REPEAT

LEAVE & ITERATE

The LEAVE statement is the same as the Break keyword in other programming languages such as Java. The Leave statement is used to exit the loop immediately without waiting for checking the condition. The use of LEAVE is shown in the following example inside a WHILE loop. Statement1 is executed only once for the case when the expression evaluates to true. This is because the LEAVE statement breaks the loop without even executing Statement2.

WHILE expression DO
  Statement1 ;
  LEAVE ;
  Statement2 ;
END WHILE

The ITERATE statement is used to skip the entire code under it and skip to the next iteration. The ITERATE statement is similar to the Continue statement in PHP and Java. For instance for the following WHILE loop example, Statement1 is executed always as long as the expression evaluates to true. However, Statement2 is never executed as it is under the ITERATE statement which forces the program to always skip to the next iteration without completing the execution of the code below the ITERATE statement.
6.1.5 Cursors

SQL Cursors are used to handle a result set retrieved by a query inside a stored procedure whereby the returned result sets are iterated row by row. In MySQL, the data returned by a cursor is read only, non-scrollable and asensitive. An asensitive cursor points normally to the actual data itself whereas an insensitive cursor points to a temporary copy of the data.

First, to create a cursor, you need to declare a cursor by using the DECLARE statement as shown in the following syntax:

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
```

The cursor declaration must be after all variable declarations as if you declare a cursor before variables declaration, MySQL would raise an error. A cursor should always use a SELECT query to retrieve a result set. For instance, to declare a cursor named `curs_emails_names` to get all emails and names of clients, we use:

```
DECLARE curs_emails_names CURSOR FOR
    SELECT name, email FROM clients;
```

The next step is to open the cursor using the OPEN statement. The OPEN statement initializes the result set for the cursor. To open the declared cursor named `curs_emails_names`, we use the following statement:

```
OPEN curs_emails_names ;
```

To start reading and processing rows from the result set, the FETCH statement is used to retrieve the next row pointed by the cursor and move the cursor to the next row within the result set as illustrated with the following syntax:

```
FETCH cursor_name INTO variable1 [, variable2] ... ;
```
For example, to process a row and store its two attribute values into two variables named `vname` and `vemail`, we use:

```
FETCH curs_emails_names INTO vname , vemail ;
```

Once done, the CLOSE statement needs to be called to deactivate the cursor and release the memory associated with it as follows:

```
CLOSE curs_emails_names ;
```

To show the use of cursor within MySQL, we will create a procedure to return the email list for the clients formatted in the following structure:

```
ClientName1<email1@gmail.com> ;
ClientName2<email2@yahoo.com> ;...
```

The name of the table is `clients` that has a number of attributes including `name` and `email`. The stored procedure is called `buildEmailList` which takes an OUT parameter used for returning the mailing list. The complete code for the stored procedure is given below as:
DELIMITER //

CREATE PROCEDURE buildEmailList (INOUT list TEXT)
BEGIN

    DECLARE vname varchar(255) DEFAULT "";
    DECLARE vemail varchar(255) DEFAULT "";

    DECLARE vfinished INTEGER DEFAULT 0;
    DECLARE my_cursor CURSOR FOR
        SELECT name, email FROM clients;

    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET vfinished = 1;

    OPEN my_cursor;

    WHILE vfinished=0 DO
        FETCH my_cursor INTO vname, vemail;
        IF vfinished = 0 THEN
            SET list=CONCAT(list,vname,'<',vemail,">;") ;
        END IF;
    END WHILE;

    CLOSE my_cursor;

END
//
DELIMITER ;

Note that we have defined a new variable called \textit{vfinished} with an initial value of 0. Meanwhile, we have declared a Handler for the type NOT FOUND as:

\begin{verbatim}
DECLARE CONTINUE HANDLER
    FOR NOT FOUND SET vfinished = 1;
\end{verbatim}

When using a cursor in MySQL, the handler NOT FOUND must be defined to deal with the situation when the cursor could not find any row. When the cursor reaches the end of the result set, it will not be able to fetch the data, and an exception is raised. The handler is used to handle this exception or condition which is like catching exceptions in
Java Language. For our case, whenever the Handler is triggered, it would assign a new value of 1 to the variable $vfinished$.

To call the procedure $buildEmailList$, we use the CALL statement as shown below with a predefined variable named $@list$:

```sql
SET @list="";
CALL buildEmailList(@list);
SELECT @list;
```

6.2 Triggers

A database trigger is procedural code which is automatically invoked and executed in response to certain events that occurred on a particular table or view within the database. The trigger is mostly used for maintaining the integrity of the information on the database as well as to ease automation of data insertion and modification. The SQL trigger which is composed usually of a set of SQL statements stored in the database catalog, is fired whenever an event such as insert, update or delete is applied on a certain table. The SQL trigger can be considered a special type of stored procedure which is invoked automatically as opposed to the normal procedure which called directly by the user or an application. The main use of benefits of using SQL triggers is outlined as follows:

- Audit changes: as to keep a log of all activities conducted by the different the users on the database.
- Enhance changes: for instance, for every change performed on the database, we add a time-stamp to the updated record.
- Enforce business rules: as to require that every invoice have at least one product.
- Execute business rules: For instance to notify the user whenever their number of products within the stock is approaching to reach zero.
- Replicate data: as to clone every newly inserted row into another table for backup.
- Enhance performance: For instance to compute the total balance after every transaction for faster access.

6.2.1 Creating Triggers in MySQL

In MySQL, a trigger is a set of SQL statements that is called or fired automatically when a change is made to the data on the associated table. The trigger can be defined to be invoked either before or after the data is changed by one of the following SQL statements:
INSERT, UPDATE or DELETE. Therefore, a maximum of six triggers can be defined for each table which are listed as follows:

- **BEFORE INSERT**: invoked before new data is inserted into the table.
- **AFTER INSERT**: invoked after new data is inserted into the table.
- **BEFORE UPDATE**: invoked before data in the table is changed.
- **AFTER UPDATE**: invoked after data in the table is changed.
- **BEFORE DELETE**: invoked before data is deleted from the table.
- **AFTER DELETE**: invoked after data is deleted from the table.

In order to create a trigger, the statement `CREATE TRIGGER` is used as illustrated using the following syntax. In the same way for procedure, delimiter is recommended to be used to avoid interrupting the DBMS by the semi-colon (;):

```sql
DELIMITER //
CREATE TRIGGER trigger_name
  [BEFORE or AFTER] [INSERT or UPDATE or DELETE]
  ON table_name
  FOR EACH ROW
  BEGIN
      ...
  END
//
DELIMITER ;
```

The keyword **BEFORE** is used when you want to process action prior to the change is made on the table meanwhile, the keyword **AFTER** should be used to process an action after the change is made. The trigger event can be **INSERT**, **UPDATE** or **DELETE**. A trigger only can be invoked by one event. To define a trigger that is invoked by multiple events, you have to define multiple triggers, one for each event or instead define a procedure that can be invoked from the trigger. More importantly when writing the body for a trigger, is to remember the use of **OLD** and **NEW** for accessing old or new data to be modified as shown in the examples within the next section.

To retrieve the list of triggers in a particular database named *abc*, you can executing the following SQL statement:
SELECT * FROM Information_Schema.Trigger
  WHERE Trigger_schema = 'abc';

To remove an existing trigger from the database, the DROP TRIGGER statement is used as follows to remove a trigger named `hello` attached on a database table called `client`:

DROP TRIGGER client.hello;

### 6.2.2 Examples for SQL Triggers

To show an audit example of where we log all update activities made a table called `clients`. Update activities are stored into a new table called `client_audit` where we store the old name (OLD.name) as well as the new inserted name (NEW.name) along with the timestamp of the update.

```sql
DELIMITER //
CREATE TRIGGER before_client_update
  BEFORE UPDATE ON clients
  FOR EACH ROW
  BEGIN
    INSERT INTO client_audit
      (oldname, newname, changedtime) VALUES
      (OLD.name, NEW.name, NOW());
  END
//
DELIMITER ;
```

To create an SQL trigger to check that an amount paid must be positive. For the case where the newly inserted amount is negative, the trigger should assign it a new value of 0 instead of the negative value. The trigger named `check_amount_before` attached to the table `payment` is invoked before the insert event as shown below:
DELIMITER //
CREATE TRIGGER check_amount_before
BEFORE INSERT ON payment
FOR EACH ROW
BEGIN

  IF NEW.amount<0 THEN
    NEW.amount=0 ;
  END IF;

END
/
DELIMITER ;

Finally, to illustrate the use of SQL triggers for enhancing performance, we will create a trigger to compute the total amount paid after every order paid by the client. The total amount paid is stored within an attribute named totalpaid of the clients table for faster access. The trigger named compute_balance attached to the table orders is invoked after any changes within the orders table including insert, delete and update events. To do this, we will first create a procedure named proc_compute_balance which takes the clients_id as an IN parameter outlined below:

DELIMITER //
CREATE PROCEDURE proc_compute_balance(IN cid)
BEGIN
  DECLARE total_balance INT DEFAULT 0 ;

  SELECT SUM(price) INTO total_balance
  FROM orders WHERE clients_id=cid ;

  UPDATE clients SET
  totalpaid=total_balance
  WHERE clients_id=cid ;

END
/
DELIMITER ;

We need now to create three trigger and attach them to the table orders as followed. This is mainly because MySQL does not allow for multiple events on the same trigger. Note that
the same pieces and statements of code explained earlier within the procedure section can be employed when creating a trigger such as variable declaration, SELECT INTO statement and cursors.

```sql
DELIMITER //
CREATE TRIGGER compute_balance_insert
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
    CALL proc_compute_balance(NEW.clients_id);
END
//
DELIMITER ;
```

The trigger for the delete event should be as given below:

```sql
DELIMITER //
CREATE TRIGGER compute_balance_delete
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
    CALL proc_compute_balance(NEW.clients_id);
END
//
DELIMITER ;
```

The trigger defined for the update event is outlined below:

```sql
DELIMITER //
CREATE TRIGGER compute_balance_update
AFTER UPDATE ON orders
FOR EACH ROW
BEGIN
    CALL proc_compute_balance(NEW.clients_id);
END
//
DELIMITER ;
```
6.3 Exercises & Questions

Try to answer the following questions

1. What’s the difference between a procedure and a trigger?
2. How to list existing procedures in a database?
3. Why do we need to declare the NOT FOUND handler within a trigger?
4. What’s the difference between : OUT and INOUT?
5. What’s the difference between : WHILE and REPEAT loops?
6. What are the triggers that can be attached to a table?
7. What are the main benefits of using triggers?

Given the logical structure of the following four tables:

```sql
clients(clients_id, countries_id, name, last_name)
countries(countries_id, country)
products(products_id, name, price)
orders(orders_id, clients_id, products_id, price, o_time)
```

The `o_time` is of type DATE with the format (Ex: '2012-11-07')

8. Write an SQL procedure to return the total sale for a given country.
9. Write an SQL procedure to return the difference of sale between two given years.
10. Suppose that we want to update the `clients_id` for client. Write an SQL trigger to automatically update the corresponding `clients_id` under the `orders` table. This is because we have not used Foreign keys here.
11. In the same way as above, if we delete a client, we need to delete their orders. Write an SQL trigger to do this.

Bibliography


Chapter 7
Object Relational Databases

7.1 Introduction to ORDBMS

Object Relational Database Management System (ORDBMS) is a database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. In addition, just as with proper relational systems, it supports extension of the data model with custom data-types and methods. The main benefit to this type of database lies in the fact that the software to convert the object data between a RDBMS format and object database format is provided. Therefore it is not necessary for programmers to write code to convert between the two formats and database access is easy from an object oriented computer language.

ORDBMS puts an object oriented front end on a RDBMS. When an application communicates with an ORDBMS it will normally act as though the data is stored as objects. Then the ORDBMS will convert the object information into data tables with rows and columns and handle the data as it were stored in a RDBMS. The main features for Object-relational database system are summarized as follows:

- Complex data: Because new data types can be created therefore any complex data can be modelled using the creation of new types. Complex data creation in most SQL ORDBMSs is based on preliminary schema definition via the user-defined type (UDT)

- Inheritance: a structured type can have subtypes that reuse all of its attributes and contain additional attributes specific to the subtype

- Method Encapsulation: Objects stored inside ORDBS can have their own methods to perform various operations. As database tables contain only data. Objects can hold data as well as include functions to perform operations that are likely to be performed on that data.
Even though RDBMS and ORDBMS are both DBMSs, they differ in how they interact with applications. Applications using RDBMS has to do extra work when storing complex data while ORDBMS inherently provide support for this. But due to the internal conversion between data formats, performance of ORDBMSs can be degraded. Therefore choosing one over the other is dependent on the data that needs to be stored/managed.

7.2 Creating New Types

The main benefits of Oracle of other ORDBMS is the ability to model complex data through the creation of newly user defined types (UDT). For instance, we can create a new type for the Point as a new class. The point should have two attributes or instance variables that are the $x$ and $y$ coordinates of a primitive type NUMBER. The CREATE TYPE statement is used to create a new type as shown below:

```sql
CREATE TYPE Point AS OBJECT (
    x NUMBER,
    y NUMBER
);
```

Newly defined types can be used as attributes to create new types. For instance, we can create new type named Side which is usually composed or represented by two points. Therefore, the Side type should have two attributes as of type Point as shown in the statement below:

```sql
CREATE TYPE Side AS OBJECT (
    a Point,
    b Point
);
```

7.3 Handling Objects

7.3.1 Object Tables

In the same way as for relational database management system, tables can be created using the same SQL syntax including the use of primary keys, different primitive data types and so on. For instance, to create a table named mytable having two attributes of types VARCHAR and Point respectively, we can use the following SQL statement:

```sql
CREATE TYPE Point AS OBJECT (
    x NUMBER,
    y NUMBER
);
```

Newly defined types can be used as attributes to create new types. For instance, we can create new type named Side which is usually composed or represented by two points. Therefore, the Side type should have two attributes as of type Point as shown in the statement below:

```sql
CREATE TYPE Side AS OBJECT (
    a Point,
    b Point
);
```

```sql
CREATE TYPE Side AS OBJECT (
    a Point,
    b Point
);
```
CREATE TABLE mytable (  
  PointName VARCHAR2(200),  
P     Point,  
);  

However, In Object-Relational database, we can create a special table to store only objects of the same type as shown in the following SQL statement whereby we create a table of objects named mypoints to store only objects of type Point:

CREATE TABLE mypoints OF Point;

### 7.3.2 Inserting Objects

To insert objects into a table, we usually form an instance of the object using its constructor. The parameter for the constructor are its attributes or instance variables. For instance, to insert a Point object into the table of objects already created as mypoints, the Point object needs two parameters as its two defined attributes \( x, y \). The SQL statement to insert a point having the cooradinates of \((1,2)\) is given below as:

```sql
INSERT INTO mypoints VALUES ( Point('1', '2') );
```

In the same way as Object-Oriented Programming, we can use the `NEW` keyword as shown below:

```sql
INSERT INTO mypoints VALUES ( new Point('1', '2') );
```

With some reserve and caution, the above can be shortened to the following, but highly discouraged.

```sql
INSERT INTO mypoints VALUES ( '1', '2' );
```

To show a further complex example, we initially create an object table named mysides for storing objects of type Side as shown in the following SQL statement:

CREATE TABLE mysides of side;

The insertion into the table mysides would be given as follows:
Chapter 7. Object Relational Databases

```java
INSERT INTO mysides VALUES
(new Point('1', '2'), new Point('4', '5'));
```

The following insertion statement is not guaranteed to work for you:

```sql
INSERT INTO mysides VALUES
('1', '2', '4', '5');
```

### 7.3.3 Querying Objects

To search for objects or data within a table, we usually use the SELECT statement along with other SQL clauses such as WHERE, ORDER BY and GROUP BY. For instance, to list all `Point` objects stored into the table named `mypoints` we use the following SQL statement:

```sql
SELECT * FROM mypoints;
```

In the same way as regular relational table, we can list or display the list of attributes defined within the `Point` class using the Dot notation which is similar to Java Programming Language. For example, to list the `x` and `y` coordinates for points stored under the table `mypoints`, we use:

```sql
SELECT p.x, p.y FROM mypoints p;
```

Note that under Oracle, Aliases (AS under MySQL) is usually achieved just be writing the alias after the object or table name without using AS. For instance, the `p` is an alias for the table `mypoints`. Let us now query the table `mysides` that stores the object of type `Side`.

Using the following SQL statement:

```sql
SELECT * FROM mysides;
```

The above SQL using the wildcard operator * would usually works if you are accessing Oracle from Java or other programming languages, but if you are trying directly, it will not work for you. Instead, try to following query :

```sql
SELECT s.a.x, s.a.y FROM mysides s;
```

The `s.a.x` means the coordinate `x` belonging to object `a` which is of type `Point`. `a` belongs to the object `s` which is an alias for `mysides` of type `Side`. 
7.3.4 REF Pointers

Every row in an object table is assigned a unique object identifier (OID), which is a hexadecimal number generated by Oracle. An object reference, represented using an Oracle REF type, can store an OID and is used as a pointer to a row in an object table. An object reference is similar to a pointer in the C programming language and is the object table version of a foreign key. To further outline the use of REF or pointers within Oracle, let’s suppose we have a Lecturer object created using the following statement:

```
CREATE TYPE lecturer AS OBJECT (
    name VARCHAR2(100),
    researchArea VARCHAR2(100)
);
```

We create an object table called `tb_lecturers` to store the list of existing lecturers within the school.

```
CREATE TABLE tb_lecturers OF Lecturer;
```

Let’s create new type for the department with the school or university called: Department which has two attributes: name and director

```
create type Department as object (
    name VARCHAR2(100),
    director REF Lecturer
)
```

Note for the use of REF to mean that director attribute points or references an existing object of the type Lecturer. Let’s create an object table for storing departments as given below:

```
CREATE TABLE tb_departments OF Department;
```

Let’s insert a new Lecturer into our database with the name Nouzha and research topic Agents as shown below:

```
INSERT INTO tb_lecturers VALUES
    (new Lecturer('Nouzha', 'Agents'));
```
We insert now a new department for *Computer Science* and having the Lecturer named *Nouzha* as its director. The SQL code for inserting such department would potentially look as follows for the case where we don't use REFS within the creating of the Department type.

```sql
INSERT INTO tb_departments VALUES
  (
    new Department(
      'Computer Science',
      new Lecturer('Nouzha', 'Agents')
    )
  )
```

The question now, is the Lecturer object of the name *Nouzha* inserted into the table *tb_lecturers* is the same object just inserted within the *Computer Science* department? They are not the same object though they have the same value, but within Oracle they are totally different objects pointing at different memory addresses. If we modify one object as we change the research topic of *Nouzha* object within the lecturer table to "Databases", the other object would stay as it is!

Because we have used REF when creating the department object, the correct SQL code for inserting the department for *Computer Science* is given below as:

```sql
INSERT INTO tb_departments VALUES
  (
    new Department(
      'Computer Science',
      ( SELECT REF(p) FROM tb_lecturers p WHERE
        p.name='Nouzha' AND ROWNUM=1)
    )
  )
```

We have used the SQL SELECT statement to retrieve a reference (REF) for the Lecturer object named *Nouzha*. The operator ROWNUM is the same as the LIMIT function within MySQL utilised to return the first row.

In case of a SELECT query on the *tb_departments* table, we can use the DEREF operator to get an actual object data from a reference or pointer to an object as shown in the example below to find the name of the director for the Computer Science department:
7.4 Methods

Object methods are functions or procedures which are declared along with an object type definition. Methods implement usually the behavior that you want objects of that type to perform when invoking these methods from an application. The methods or functions within Oracle can be written in PL/SQL or Java programming language. There are different types of methods that can written in Oracle which are outlined in this section.

7.4.1 Constructor Methods

In the same way as in Java, a constructor method is a function which is used to create a new instance of the user-defined type taking a set of arguments which are used to set up the values of its attributes. To invoke the constructor method to create an object, the keyword NEW can be used, but is not required. Constructor methods are usually invoked when inserting new objects. For instance, to insert a new object of type Side into the table mysides, the following SQL statement is used.

\[
\text{INSERT INTO mysides VALUES ( NEW Side( NEW Point('1', '2'), NEW Point('4', '5')) );}
\]

7.4.2 Member Methods

Member methods which are called in Java instance methods as they belong to the actual object in a way they provide an application with access to the data for the actual object instance. The signatures for member methods must be first defined within the type definition using the keyword MEMBER FUNCTION along with any parameters and return value without the actual code implementation. For example, the object Side has a member function named \text{get_length()} which takes no argument and return an Integer as shown in the following SQL statement:

\[
\text{SELECT DEREF(d.director).name FROM tb_deptartments d WHERE d.name='Computer Science'}
\]
CREATE OR REPLACE TYPE Side AS OBJECT(  
a Point,
b Point,
MEMBER FUNCTION get_length() RETURN INTEGER);

To have the code implementation for the member methods, you can write it using PL/SQL language inside the CREATE TYPE BODY statement. Functions are the same as SQL procedures explained in the previous chapter. The method implementation should be enclosed between the BEGIN ... END block. For the get_length() function declared within the Side object, is implemented as given below as:

CREATE OR REPLACE TYPE BODY Side AS
  MEMBER FUNCTION get_length RETURN integer IS
    BEGIN
      RETURN sqrt( ((self.a.x-self.b.x)*(self.a.x-self.b.x)) + 
                    ((self.a.y-self.b.y)*(self.a.y-self.b.y)));
    END;
  END;
END;

Note that the use of self to refer to the object iself which is the equivalent to this in Java.
To invoke the member method for an object, the dot notation is used as shown in the following example for a simple SELECT statement with WHERE clause.

SELECT p.get_length() FROM mysides p WHERE p.get_length() > 10 ;

7.4.3 Static Methods

Static methods are invoked on the object type or class but not its instances as in Java Language. Static methods are employed for operations that are global to the type and therefore do not need to reference or access the data of an object instance. Static methods must be declared within the type definition as member methods using STATIC FUNCTION or STATIC PROCEDURE keywords as shown for the Point type where a static function named info is defined. The method info takes two number parameters and returns a string value.
CREATE TYPE Point AS OBJECT(
  x NUMBER,
  y NUMBER,
  STATIC PROCEDURE info (a NUMBER,b NUMBER) RETURN STRING
);

To implement the code for the static function `info` which returns a simple string for two given numbers, the function must be placed within the BODY for the type. Note that all static and member methods should be placed within the BODY block. The `||` is the concatenation operator of string variables within Oracle.

```sql
CREATE OR REPLACE TYPE BODY Point AS
  STATIC PROCEDURE info(a number, b number) RETURN string IS
    BEGIN
      return 'point info (' || a || ' , ' || b || ')'
    END;
  END;
END;
```

### 7.4.4 Map & Order Methods

In order to compare and sort instances of an object type, there must be a way or a basis for comparing them. For instance, scalar data type such as CHAR or INT have a predefined order based on their values to be compare them. But a user defined object type, such as a `Side`, which can have multiple attributes of various data types, has no predefined axis or basis of comparison. Oracle has the option to define a `map` method or an `order` method for comparing objects, but **not both**. An order method compares the values of an instance against a given instance of the same time meanwhile a map method returns a scalar values for an instance so that instances can be ordered or sorted by their position on the scalar axis. Both methods are further explained in this section.

**Order**

Order methods make direct one-to-one object comparisons. They simply tell you that the current object is less than, equal to, or greater than the object that it is being compared to, based on the criterion used. For instance, the `Side` object has an order method named `mymatch` which must takes as argument an objet of the same type (`Side`) and returns a number of either value: 0, 1 or -1. The order method should be declared within the type definition using the ORDER MEMBER FUNCTION whilst it must be implemented within the BODY part of the object.
CREATE OR REPLACE TYPE Side AS OBJECT(
    a Point,
    b Point,
    MEMBER FUNCTION get_length() RETURN INTEGER
    ORDER MEMBER FUNCTION mymatch(s Side) RETURN NUMBER
);

The implementation for the order method is given below where the function returns 0 is they are the same. If the actual instance is greater than the given object as argument then 1 must be return otherwise -1 should be returned. The comparison is based on the length of the side.

CREATE OR REPLACE TYPE BODY Side AS
    ORDER MEMBER FUNCTION mymatch(s Side) RETURN NUMBER IS
        BEGIN
            IF self.get_length() < s.get_length() THEN
                RETURN -1;
            ELSIF self.get_length() > s.get_length() THEN
                RETURN 1;
            ELSE
                RETURN 0;
            END IF;
        END;
    MEMBER FUNCTION get_length RETURN integer IS
        BEGIN
            ...
        END;
END;

For instance, to retrieve all objects which are greater than a given Side, we use the special function VALUE() as shown below:

SELECT * FROM tb_sides s WHERE
    VALUE(s) > Side(Point(1,2), Point(3,5))

Map
Map method is used to return a scalar value for an instance which is used rank or sort the instances of the same type on a given scalar axis. For instance, the Side object has a map method declared using the MAP keyword named mymap as shown below:
CREATE OR REPLACE TYPE Side AS OBJECT(
    a Point,
    b Point,
    MEMBER FUNCTION get_length() RETURN INTEGER
    MAP MEMBER FUNCTION mymap RETURN NUMBER
);

The implementation of the map method `mymap` should just return the length of the side:

CREATE OR REPLACE TYPE BODY Side AS
    ORDER MEMBER FUNCTION mymap() RETURN NUMBER IS
    BEGIN
        RETURN self.get_length();
    END;
    MEMBER FUNCTION get_length RETURN integer IS
    BEGIN
        ....
    END;
    END;

To have a SELECT statement to get all objects stored into table `tb_sides` and order them. The ordering mechanism is based on the MAP function invoked through the use of the VALUE() function.

SELECT * FROM tb_sides s ORDER BY VALUE(s)

7.5 Inheritance

Inheritance is the mechanism that connects sub-types in a hierarchy to their supertypes (parents). Subtypes automatically inherit the attributes and methods from their parent type. Note that Oracle supports only single inheritance as opposed to C++ programming. Therefore, a subtype can derive directly from only one parent type. The main keyword within Oracle to have inheritance from the supertype is UNDER. For instance, we create a new type named `Person` as shown below. The type must be declared using the statement NOT FINAL so that it can inherited as Oracle defines all new types as FINAL by default.
CREATE TYPE Person AS OBJECT(
   name VARCHAR2(20)
   age NUMBER) NOT FINAL;

To create a subtype named \textit{Student} which is based on the parent type \textit{Person}:

\begin{verbatim}
CREATE TYPE student UNDER person(
   school VARCHAR2(100)
);
\end{verbatim}

Note that the new type \textit{Student} should have an attribute named \textit{school} in addition to other two inherited attributes from the parent type which are \textit{name} and \textit{age}. The type \textit{Student} can not have further children types as it is not declared NOT FINAL. Let us create an object tabled named \textit{tb\_persons} to store objects of type \textit{Person}.

\begin{verbatim}
CREATE TABLE tb\_persons AS person;
\end{verbatim}

The object table can be used to store all objects whose their type \textit{Person} as well as subtypes under \textit{Person} as shown below using the SQL insert statements

\begin{verbatim}
INSERT INTO tb\_persons ( Person('Imed',67) );
INSERT INTO tb\_persons ( Student('Asma',7) );
\end{verbatim}

That’s for attributes. For methods, subtypes do also inherits all declared methods within a parent type. Furthermore, a subtype can override the inherited methods to provide a different implementation. For example, the parent type \textit{Person} have two member methods \textit{getinfo} and \textit{getage} as shown below:

\begin{verbatim}
CREATE TYPE Person AS OBJECT(
   name VARCHAR2(20)
   age NUMBER,
   member function getinfo return VARCHAR2(100),
   member function getage return VARCHAR2(100),
) NOT FINAL;
\end{verbatim}

The implementation of the functions which return the name as well as the age of the person with string prefixes, are given below:
CREATE OR REPLACE TYPE BODY Person AS
MEMBER FUNCTION getinfo RETURN varchar2(100) IS
BEGIN
    RETURN 'Name ' || self.name;
END;
MEMBER FUNCTION getinfo RETURN varchar2(100) IS
BEGIN
    RETURN 'Age is ' || self.age;
END;
END;

For the subtype Student inheriting the Person should be inheriting all attributes and methods. However, To override the implementation of an inherited method, such method must be declared using the keyword OVERRIDING as shown in the following SQL code:

```
CREATE TYPE Student UNDER Person(
    school VARCHAR2(100),
    OVERRIDING MEMBER FUNCTION getinfo RETURN VARCHAR2(100),
    member function getschool return VARCHAR2(100),
);
```

The implementation of the overriding method for the subtype Student is given as follows:

```
CREATE OR REPLACE TYPE BODY Student AS
OVERRIDING MEMBER FUNCTION getinfo RETURN varchar2(100) IS
BEGIN
    RETURN 'Student Name ' || self.name;
END;
MEMBER FUNCTION getschool return VARCHAR2(100) IS
BEGIN
    RETURN 'School ' || self.school;
END;
END;
```

Note that instances of the subtype are also instances of the parent type. When querying the table tb_persons, all instances are considered as Person:

```
SELECT p.getinfo(), p.getage() FROM tb_persons p
```
The SELECT statement would display the following results observing that the subtype Student returns results differently from the Person instance because of the overriding of the method.

<table>
<thead>
<tr>
<th>Name</th>
<th>Imed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student Name</td>
<td>Asma</td>
</tr>
</tbody>
</table>

### 7.6 Collections

Oracle provides two techniques for modeling one-to-many relationships. This is relevant for the case where an attribute contains a set or a collection of values. Oracle Collections can be modelled as VARRAY or Nested tables in which a collection can be columns in tables or attributes of object types.

#### 7.6.1 VARRAY

VARRAY are like a C array used to store a fixed number of instances which must be of the same type. Each instance stored within the VARRAY has a numerical index that can be used for accessing the instance. The simple syntax for creating a VARRAY is given as:

```sql
CREATE TYPE type_name AS VARRAY (limit) OF data_type;
```

For example, let us suppose we have the following types being created; `Point` and `Side`.

```sql
CREATE TYPE Point AS OBJECT(
    x NUMBER,
    y NUMBER);
/
CREATE TYPE Side AS OBJECT(
    a Point,
    b Point);
```

To create a VARRAY named `fourSides` of type `Side` with a limit of 4 instances:

```sql
CREATE TYPE fourSides AS VARRAY(4) OF Side;
```

The VARRAY can be an attribute (column) for a table or an attribute within a type. For instance, we have created a table named `Square` where the VARRAY is set as one of its attributes.
CREATE TABLE Square (  
    name VARCHAR2(20),  
    sd fourSides  
);  

In order to initialize an instance of the VARRAY during the insert statement, we use the constructor as shown below:

INSERT INTO Square VALUES (  
    'Square One',  
    fourSides(  
        side(Point(1,2), Point(2,3)),  
        side(Point(3,2), Point(4,3)),  
        NULL  
    )  
);  

For a VARRAY, you do not have to set all the values. To retrieve data using SELECT, you are advised to use PL/SQL. Though the function TABLE can be used to flatten the values within a collection as shown below:

SELECT c.name, b.a.x FROM Square c, TABLE(c.sd) b  

7.6.2 Nested Tables

Nested Table is just a table that can be stored inside a table or a new type as an attribute. As opposed to VARRAY, nested tables do not have a fixed maximum size. The nested tables can have triggers and indexes but they cannot be directly queried. The syntax for creating a nested table is given below as:

CREATE TYPE table_name AS TABLE OF object_type  

For instance, to create a nested table named \( tb\_sides \), we use the following statement:

CREATE TYPE tb_sides AS TABLE OF Side  

To create a table named \( Polygon \) with a nested table as one of its attribute, the following SQL statement is used:
CREATE TABLE Polygon(
    name varchar2(100),
    sd  tb_sides
)NESTED TABLE sd STORE AS sd;

Note that you must declare explicitly at the end of any table that contains a nested table inside them directly or indirectly using the statement:

NESTED TABLE nested_table_name STORE AS any_other_given_name;

To Insert for Nested Table VARRAY:

INSERT INTO Polygon VALUES ('Square One',
    tb_sides(
        side(Point(1,2), Point(2,3)),
        side(Point(3,2), Point(4,3))
    )
);

To retrieve data using SELECT, you are advised to use PL/SQL

select c.name, b.a.x from square c, table(c.sd) b

7.7 PL/SQL

PL/SQL stands for Procedural Language as an extension of SQL. PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90’s to improve the capabilities of SQL. PL/SQL is not a stand-alone programming language; it is a tool within the Oracle programming environment. SQL* Plus is an interactive tool that allows you to type SQL and PL/SQL statements at the command prompt. These commands are then sent to the database system for processing. Once the statements are processed, the results are sent back and displayed on screen. The basic syntax for a PL/SQL program which is almost the same as the way we create procedures within MySQL is given below as:
DECLARE
    <declarations section>
BEGIN
    <executable command(s)>
EXCEPTION
    <exception handling>
END;

7.7.1 Variables

PL/SQL programming language allows to define various types of variables for storing data. The syntax for declaring a variable is:

name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT init_val]

The keyword CONSTANT is to declare a variable as a final variable whose value cannot be changed. The assignment operator within Oracle is ":=". For instance, to define a set of variables of different types within Oracle, the following PL/SQL is used. Not that it is not mandatory to set the initial value for a variable. The special function dbms_output.put_line() is the same as System.out.println() in Java which used for displaying messages on the screen.

DECLARE
    a integer := 10;
    b string := "Hello World";
    pi CONSTANT real := 3.14;
    f real;
BEGIN
    f := a + pi;
    dbms_output.put_line('Value of f: ' || f);
END;

In the same way as MySQL procedures, the SELECT INTO statement can be used to assign values from database tables into variables as shown in the following example:
7.7.2 Flow Control

The IF - ELSEIF - ELSE decision-making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a set of statements to be executed based on the given conditions. The simple syntax for the flow control within Oracle is outlined in the following SQL code:

```
IF (boolean_expression_1) THEN
    statement1;
ELSIF (boolean_expression_2) THEN
    statement2;
ELSE
    statement3;
END IF;
```

The previous example given for MySQL procedures, is rewritten for PL/SQL. Note that the comparison operator for Oracle is "==" as opposed to "=" used for MySQL.

```
IF status == 0 THEN
    SELECT count(*) INTO result FROM clients
        WHERE confirm = 'NO';
ELSIF status == 1 THEN
    SELECT count(*) INTO result FROM clients
        WHERE confirm = 'YES';
ELSE
    result := 0;
END IF
```

7.7.3 Loops

The Loop structure allows us to execute a set of statements multiple times based on a given condition. PL/SQL offers different types of loop structures including mainly WHILE and FOR. The syntax for a while statement is given as:

```
IF status == 0 THEN
    SELECT count(*) INTO result FROM clients
        WHERE confirm = 'NO';
ELSIF status == 1 THEN
    SELECT count(*) INTO result FROM clients
        WHERE confirm = 'YES';
ELSE
    result := 0;
END IF
```
WHILE condition_statement LOOP  
    set_of_statements  
END LOOP;

For example, the following loop would display on the screen the incremented value of a declared variable 10 times:

```sql
DECLARE
    a number := 10;
BEGIN
    WHILE a < 20 LOOP
        dbms_output.put_line('value of a: ' || a);
        a := a + 1;
    END LOOP;
END;
```

Meanwhile, the structure for the FOR loop is shown below:

```sql
FOR counter IN initial_value .. final_value LOOP  
    set_of_statements;
END LOOP;
```

For instance, the previous WHILE loop, is rewritten using a FOR loop to display the value of 10 times as:

```sql
DECLARE
    a number;
BEGIN
    FOR a in 10 .. 20 LOOP
        dbms_output.put_line('value of a: ' || a);
    END LOOP;
END;
```

### 7.7.4 Cursors

SQL Cursor are used to handle a result set retrieved by a query inside a stored procedure whereby The returned result sets are iterated row by row. The way to handle cursor within PL/SQL is almost the same as explained in the previous chapter for handling cursor within MySQL. The syntax for explicitly declaring a cursor in Oracle is given below as:

```sql
```
CURSOR cursor_name IS select_statement;

Fetching the cursor using the FETCH ... INTO statement involves accessing one row at a time and assigning values to declared variables as shown below for a cursor named cur_clients:

FETCH cur_clients INTO var_cid, var_name;

Luckily, with PL/SQL, you can navigate through the cursor rows using a FOR or While loop as shown below:

DECLARE
    CURSOR cur_clients IS (SELECT * FROM clients);
BEGIN
    FOR ccc IN cur_clients LOOP
        dbms_output.put_line('Client name ' || ccc.name)
    END LOOP;
END;

7.8 Practice Oracle

To use Oracle and practice the different types of SQL commands for creating new types as well as writing PL/SQL programs, you can either download the Oracle Express Edition (XE) or Oracle Enterprise Edition based on your Operating system. We have setup a page with links to the right product for your computer which can be accessed at the link below:

http://www.learndb.com/down/oracle

Note that Oracle XE is available only for Windows 32 bit, If you have a 64-bit, you may need to install the full version of Oracle. For the case you have installed the XE edition:

1. Click on : Start \rightarrow All Programs \rightarrow Oracle database 11g Express Edition \rightarrow Go to Database Home Page as shown in Figure (7.1).

2. Your browser would be opening an admin page for Oracle to login. The password is the one you have set during the installation.

The username for login is : system
3. Once logged in inside, you will be having four options. Click on the SQL icon.

4. Afterwards, choose SQL Commands.

5. A page with a textarea to type in your SQL query and execute through clicking run.

Figure 7.1: Online Phone Directory for British Telecom

Figure 7.2: Online Phone Directory for British Telecom
7.9 Questions & Exercises

Try to have a go on the following questions:

1. What are the main features for ORDBMS?

2. Write a PL/SQL program to iterate through the elements of a VARRAY. Take the Square table example.

3. Create a nested table to store objects of type: Side.

4. Define a new type called Polygon with two functions: get_area and get_perimeter. The new type should be inheritable. The type should contain a nested table storing Sides as an attribute.

5. Implement the function of the Polygon: get_perimeter.

6. Define a sub-type named mysquare under the Polygon type and override the get_area function.

Bibliography


Chapter 8

Databases & Java Programming

8.1 MySQL with Java

MySQL is an open-source database management system used mainly for relational databases. It is considered the second mostly used open-source DBMS following SQLLite. MySQL is being deployed for many high-profile and large scale website including Facebook, youtube, twitter and Flickr. To install MySQL, you can just install EasyPHP as shown in the third chapter with a few easy steps.

As we initially developed the phonebook using flat-based storage, in this chapter, we will develop the phonebook using a database management system. The structure for the Java code is shown below with only the interaction part with the end-user.

Listing 8.1: PhoneBook.java : Phonebook code structure using Database systems

```java
import java.io.*;
import java.sql.*;
import java.util.*;

public class PhoneBook {
    public static String help_msg="Press: H Help - A Add contact - S Search - Q Exit :";
    public static void main(String[] args) {
        try{
            System.out.println("\n\n***** Welcome to MyPhone Book *****\n\n");
            Scanner s=new Scanner(System.in);

            //Database Connection code below:

            //Interaction with the user:
            for(;;){
                System.out.print("[Main Menu] "+help_msg+"\n: ");
                String command=s.nextLine().trim();
                if (command.equalsIgnoreCase("H")){
```
Database Connection

Here is part of the code to connect to the database. Note that the username for most installation is set to `root`, meanwhile the password for most windows machine is the empty string "". In case, the database management system is installed at a remote location, make sure you change the address `localhost` to the address where the DBMS is running from. The code executes an SQL query for creating a table named `contacts`.

Listing 8.2: PhoneBook.java : Java Connection to DBMS

```java
//Database Connection code below:
String userName = "root";
String password = "YourPassword";
String url = "jdbc:mysql://localhost/myphonebook";
Class.forName("com.mysql.jdbc.Driver").newInstance();
Connection con = DriverManager.getConnection(url, userName, password);
con.createStatement().executeUpdate("create table IF NOT EXISTS contacts (" +
   "id int auto_increment primary key," +
   "name varchar(45)," +
   "lname varchar(45)," +
   "phone varchar(45))");
```
8.1.1 Inserting Contacts

As opposed to the file-based storage for adding a new contact, through the use of DBMS the implementation becomes much easier. For inserting data, it is recommended to use PreparedStatement objects and afterwards setting the values for the question marks ? using the method: setObject or setInt or setString and so on.

Listing 8.3: PhoneBook.java : Inserting a new Person

```java
else if (command.equalsIgnoreCase("A")){
    System.out.print("Type in contact details in the format: name,lastname,phone\n: ");
    for(;;){
        String data=s.next().trim();
        String[] temp=data.split(",");
        if (temp.length!=3){
            System.out.println("Error, the insertion format should be in the format: firstname,lastname,phone :");
            continue;
        }
        PreparedStatement pstmt1 = conn.prepareStatement("insert into contacts (name,lname,phone) values ( ?, ?, ?)");
        pstmt1.setObject(1, temp[0]);
        pstmt1.setObject(2, temp[1]);
        pstmt1.setObject(3, temp[2]);
        pstmt1.executeUpdate();
        break;
    }
}
```

8.1.2 Searching Contacts

In order to search for a contact within a database, we usually use a SELECT query with the wildcard operator %. The Java code is given below where the results from the database is returned as ResultSet object that needs to be iterated through its next() method.

Listing 8.4: PhoneBook.java : Searching for a Person

```java
else if (command.equalsIgnoreCase("S")){
    System.out.print("Type in the name you are searching for :\n: ");
    String data=s.next().trim();
    data ="%"+data+"%";
    String q="select * from contacts where name like ? or lname like ?";
    PreparedStatement pst2=con.prepareStatement(q);
    pst2.setString(1, data);
    pst2.setString(2, data);
    ResultSet rs=pst.executeQuery();
```
8.1.3 Running the Java Application

To run the code for the Phonebook, you need to include the MySQL driver jar file within the classpath of your application. Follow the simple steps to get it configured:

1. Download the MySQL driver file from the link below:

   http://www.learndb.com/down/mysql.jar

2. Within Eclipse, click Run -> Run Configurations as shown in Figure (8.1)

   ![Figure 8.1: Configuration for Eclipse Project.](image)

3. Make sure you select your project from the left side panel. Otherwise, DOUBLE click on Java Application to appear.

4. Click on Classpath, then Click on User Entries, then click on Add External Jars. Locate the MySQL.jar file and add it.

5. That’s it. Apply and Run your project.
8.2 HSQLDB

HSQLDB is a lightweight relational database management system with size of 1 megabyte that can be used without any installation or complex configuration even from a USB drive. You can download the Jar file for the HSQLDB from the link below:

http://www.learndb.com/down/hsqldb.jar

The Java code for HSQLDB is the same as MySQL except for part of making the connection to the DBMS which is given below:

Listing 8.5: PhoneBook.java : Java Connection to DBMS

```java
//Database Connection code below:
String userName = "SA";
String password = "";
String url = "jdbc:hsqldb:myphonebook";
Class.forName("org.hsqldb.jdbc.JDBCDriver");
Connection con = DriverManager.getConnection(url, userName, password);
con.createStatement().executeUpdate("SET DATABASE SQL SYNTAX MYS TRUE");
con.createStatement().executeUpdate("create table IF NOT EXISTS contacts (" +
   "id int auto_increment primary key," +
   "name varchar(45)," +
   "lname varchar(45)," +
   "phone varchar(45))");
```

8.3 Oracle with Java

Here are the steps that you need to follow to write a simple Java code that connects to a database:

1. Open your Apex for Oracle: Click on Start -> All Programs -> Oracle database 11g Express Edition -> Go to Database Home Page
2. Visit the SQL Command : SQL -> SQL Command
3. Execute the following SQL to create a table called friends.

```sql
create table friends(
    fname VARCHAR2(100),
    lname VARCHAR2(100)
);
```
4. Create a new Java Project called: HelloOracle.

5. Create a new Java Class named Hello under the project HelloOracle.

6. Type the following code for the class HelloOracle

   Listing 8.6: HelloOracle.java: Java code for connecting to Oracle database

   ```java
   import java.io.*;
   import java.sql.*;
   import java.util.*;

   public class Hello {
      public static void main(String [] args){
         try{
            String userName = "system";
            String password = "YourPassword";
            String url = "jdbc:oracle:thin:@localhost:1521:xe";
            Class.forName ("oracle.jdbc.OracleDriver").newInstance ();
            Connection conn = DriverManager.getConnection (url, userName, password);
            //Inserting data:
            PreparedStatement pstmt = conn.prepareStatement("insert into friends
               (fname,lname) values ('Imed', 'Bouch')");
            pstmt.executeUpdate();
            pstmt = conn.prepareStatement("insert into friends (fname,lname) values
               ('Asma', 'Bouch')");
            pstmt.executeUpdate();
            //Searching for Data.
            pstmt = conn.prepareStatement("select * from friends");
            ResultSet rs = pstmt.executeQuery();
            while(rs.next()){
               String fname= rs.getString("fname");
               String lname= rs.getString("lname");
               System.out.println(fname+"\t\t\t"+lname);
            }
            rs.close();
            pstmt.close();
        }
        catch(Exception e){
           e.printStackTrace();
        }
    }
}
```

7. Download the JDBC Driver for Oracle from the link below.
8. Within Eclipse, click Run -> Run Configurations as shown in Figure (8.1)

9. Make sure you select your project from the left side panel. Otherwise, DOUBLE click on Java Application to appear.

10. Click on Classpath, then Click on User Entries, then click on Add External Jars. Locate the OJDBC6.jar file and add it.

11. That’s it. Apply and Run your project.
Index

Abs, 66
Access, 6, 64
ACID, 85
After, 105
Aggregate, 63, 68
Aliases, 50
Alter, 43, 79, 90
And, 46
Argument, 95
Array, 124
As, 50
Asc, 51
Assignment, 127
Atomicity, 85, 86
Attribute, 27, 29, 33
Audit, 21, 104
Auto-Commit, 90
Auto-Increment, 42, 56
AVG, 63

BaseX, 6
Before, 105
Begin, 94
Between, 49
Bigint, 41
Binary, 29
Blob, 41
Body, 94
Boolean, 41
Break, 100
Business Objects, 22
Business rule, 104
Business Rules, 22
Call, 94
Candidate key, 42
Cardinality, 27, 31, 33
Cascade, 80
Case, 98
char, 41
Check, 43
Chen, 31
Close, 102
Cloud, 10
Collection, 124
Complex data, 111
Composite, 29
Concat, 65, 103
Conceptual, 20, 26
Concurrency, 3, 5
Condition, 97
Conflict, 86
Consistency, 85
Constant, 127
Constraint, 22, 43, 77
Constructor, 117
Continue, 100
Create, 40, 75, 94, 112
Create User, 53
Create View, 75
Creately, 35
Crow’s foot, 31
Csv, 2
Cursor, 101, 108, 129

Data Type, 41
Database, 1, 4, 40
Intersect, 77
Into, 95
Isolation, 85, 86, 88, 90
Iterate, 100
Java, 11, 12, 55
JDK, 12
Join, 69
Key, 29, 42, 77
Last, 63
Lcase, 64
Leave, 100
Left Join, 72
Len, 64
Length, 64
Lifecycle, 19
Like, 49
Limit, 52
Lock, 5, 91
Logical, 6, 20, 26, 32
Logical Operators, 46
Loop, 99, 128
Maintenance, 21
Map, 120
MarkLogic, 6
Max, 63
MD5, 65
Method, 117
Migration, 20
Min, 63
Minute, 67
Model, 6
MongoDB, 6
Month, 67
MySQL, 6
Natural Join, 71
Nested, 68, 125
Nested Table, 125
Network, 8
New, 105, 106
NoSQL, 11
Not, 49
Not equal, 48
Not Found, 103
Not in, 49
Not Like, 49
Notation, 31
Now, 66
Null, 72
Numerical, 66
Object, 112
Object Table, 112
Object-Oriented model, 10
Object-Relational, 10
Old, 105, 106
Open, 101
Optional, 97
OQL, 11
Or, 46
Oracle, 6, 10, 43, 52, 111, 130
ORDBMS, 10, 111
Order, 119
Order by, 51
Ordinality, 31
Out, 95, 96
Parameter, 95
Performance, 104
Phantom read, 87
Phone Book, 1, 11
PHP, 1, 57, 95
phpMyAdmin, 57
Physical, 20
PL/SQL, 126
Pointer, 115
PostgreSQL, 6
Primary key, 42

I. Bouchrika, 2014
www.LearnDB.com
Learn Databases - 1st Ed.
Weak, 27
When, 98
Where, 46, 48, 63
While, 99, 128
Wildcard, 47, 49

XE, 130
XML, 11

Year, 67