Fundamentals of Databases notes

Todd Davies

December 30, 2014

Introduction

Databases are core, if largely invisible, components of modern computing architectures in both commercial and scientific contexts. The management of data has evolved from application-specific management of myriad files to organisation-wide approaches that see data as one of the most important assets of modern organisations and, as such, a key factor in their ability to compete and thrive. At this organisation-wide scale, database management systems (DBMSs) are the crucial piece of software infrastructure needed to achieve the desired results with consistent quality and robust efficiency. A modern DBMS is a thing of wonder and embodies in its internal construction and in its wide usability many advances in algorithms and data structures, programming language theory, conceptual modelling, concurrency theory, and distributed computing. This makes the study of databases a data-centric traversal of many of the most exciting topics in modern computing.

Aims

The aim of this course unit is to introduce the students to the fundamental concepts and techniques that underlie modern database management systems (DBMSs).

The course unit studies the motivation for managing data as an asset and introduces the basic architectural principles underlying modern DBMSs. Different architectures are considered and the application environments they give rise to.

The course unit then devotes time to describing and motivating the relational model of data, the relational database languages, and SQL, including views, triggers, embedded SQL and procedural approaches (e.g., PL/SQL).

The students learn how to derive a conceptual data model (using the Extended Entity Relationship paradigm), how to map such a model to target implementation model (for which the relational model is used), how to assess the quality of the latter using normalisation, and how to write SQL queries against the improved implementation model to validate the resulting design against the data requirements originally posed. For practical work, the Oracle DBMS is used.

The course unit also introduces the fundamentals of transaction management including concurrency (e.g., locking, 2-phase locking, serialisability) and recovery (rollback and commit, 2-phase commit) and of file organisation (e.g., clustering) and the use of indexes for performance.

Finally, the course unit addresses the topic of database security by a study of threats and countermeasures available. In the case of the former, these include potential theft and fraud as well as loss of confidentiality, privacy, integrity and availability. In the case of the latter these primarily include mechanisms for authorization and access control, including the use of views for that purpose. The course unit also addresses the topic of legal frameworks that give rise to obligations on the part of database professionals by introducing exemplars such as the 1995 EU Directive on Data Protection and the 1998 UK Data Protection Act.

Additional reading

Contents

1 An introduction to Database Management Systems

DataBase Management Systems (DBMS's) are a type of middleware that provide a layer of abstraction for dealing with databases. It is nearly always unnecessary to write software from scratch that interfaces with a database, since a lot of database operations will share a significant amount of logic.

Henceforth, a lot of the functionality required of applications that make use of a database is placed into a DBMS, which application developers can make use and save time. The DBMS acts as a service, that is well implemented and is able to enforce good practices and advanced techniques such as concurrency, sharding, recovery management and transactions.

Some advantages of using a DBMS include:

- It decouples data inside a database from the application using it. Either can be re-written at any time so long as they still provide/use the same interface.
- Since the data is decoupled from the application, using a DBMS (in theory) lowers the development cost of the application.
- Most DBMS are scalable, concurrent, fault tolerant, authorisation control (often role based for organisations).

Even though the DBMS aims to provide a layer of abstraction for a user application, there are several layers of abstraction within the DBMS itself. These are:

Physical	Deals with the file(s) that is written to the storage medium that will hold the database. Needs to know about file formats, indexing, compression, etc.
Logical	Mainly concerned with mapping the raw data into database 'concepts' such as tables, views etc. It is here that the formal specification of the database is defined, commonly used models include *relational, XML based and document based*
View	Ensures that only authorised people can view the data.

If the database is using a relational format, then it will be defined by a schema. A schema dictates how the database is formatted; what tables there are, and what datatypes their columns take. An instance of a database is the content (data) inside of the database at a particular point in time. There is a certain isomorphism between relational databases and imperative programming languages; a schema would be akin to the declaration of variables (i.e. their names and types), while the instance would be their values at a particular point in the program's execution.

Irrespective of what logical model a database uses, most DBMS use between one and three languages to interface with a user/application. These are:

- Data Definition Language - used for specifying schema.
- Data Manipulation Language - used for mutating the data in the database.
- Data Query Language - used to access data in the database.

Often DBMS languages will be both a DML and a DQL, and sometimes a DDL too! One such example is SQL, does all of the above!

2 Relational Algebra and SQL

Relational algebra is designed for modelling data stored in relational databases (i.e. tables) and defining queries on it. It can perform unary operations (such as growing, shrinking and selecting from tables), or binary operations (union, intersection, difference, product, join).

2.1 Selection σ

The σ operator can select rows that meet a certain criteria from the table.

Alcohol-Selection

Type	Strength	Colour
Wine	11	Red
Beer	4.2	Yellow
Wine	12.8	White
Port	18	Carmine
Ale	11	Red

If we run $\textit{Fine-Wines} := \sigma_{Type=Wine}(\textit{Alcohol- Selection})$, we'll end up with:

Fine-Wines

Type	Strength	Colour
Wine	11	Red
Wine	12.8	White

2.2 Projection π

The π operator can select rows instead of columns. If we do $\textit{Anonymous-Drinks} := \pi_{Strength,Colour}(\textit{Alcohol-Selection})$:

Anonymous-Drinks

Strength	Colour
11	Red
4.2	Yellow
12.8	White
18	Carmine

Notice that both the 11% Ale and the 11% Red Wine have the same strength and colour values. Consequently, the projection operator combines those rows into one so the same result isn't displayed twice.

Projection can also be used to do simple arithmetic, $\textit{Test} := \pi_{Strength+Strength->DStrength,Colour}(\textit{Anonymous-Drinks})$:

Test

DStrength	Colour
22	Red
8.4	Yellow
25.6	White
36	Carmine

2.3 Product $\times$

Shops

Name	Dodginess	Price
Ali's	High	Medium
Tesco	Low	Medium
New Zeland Wines	X.High	Low

We could do a cross product with the Shops and the Alcohol-Selection tables $\textit{Grog-Shops} := \textit{Shops} \times \textit{Alcohol-Selection}$:

Grog-Shops

Name	Dodginess	Price	Type	Strength	Colour
Ali's	High	Medium	Wine	11	Red
Ali's	High	Medium	Beer	4.2	Yellow
Ali's	High	Medium	Wine	12.8	White
Ali's	High	Medium	Port	18	Carmine
Ali's	High	Medium	Ale	11	Red
Tesco	Low	Medium	Wine	11	Red
Tesco	Low	Medium	Beer	4.2	Yellow
Tesco	Low	Medium	Wine	12.8	White
Tesco	Low	Medium	Port	18	Carmine
Tesco	Low	Medium	Ale	11	Red
New Zeland Wines	X.High	Low	Wine	11	Red
New Zeland Wines	X.High	Low	Beer	4.2	Yellow
New Zeland Wines	X.High	Low	Wine	12.8	White
New Zeland Wines	X.High	Low	Port	18	Carmine
New Zeland Wines	X.High	Low	Ale	11	Red

2.4 Renaming ρ

The notation for renaming columns is pretty simple; $\textit{Drinks} := \rho_{Name,Strength,Hue}(\textit{Alcohol-Selection})$

Drinks

Name	Strength	Hue
Wine	11	Red
Beer	4.2	Yellow
Wine	12.8	White
Port	18	Carmine
Ale	11	Red

2.5 Join θ

If we had:

People

Name	Drinks
Alice	Wine
Bob	Beer

We could join it with the Grog-Shops table, using $\textit{Fave-Shops} := \textit{People} \bowtie_{People.Drinks=Grog-Shops.Type} (\textit{Grog-Shops})$

Fave-Shops

Person.Name	Grog-Shops.Name	Dodginess	Price	Type	Strength	Colour
Alice	Ali's	High	Medium	Wine	11	Red
Bob	Ali's	High	Medium	Beer	4.2	Yellow
Alice	Ali's	High	Medium	Wine	12.8	White
Alice	Tesco	Low	Medium	Wine	11	Red
Bob	Tesco	Low	Medium	Beer	4.2	Yellow
Alice	Tesco	Low	Medium	Wine	12.8	White
Alice	New Zeland Wines	X.High	Low	Wine	11	Red
Bob	New Zeland Wines	X.High	Low	Beer	4.2	Yellow
Alice	New Zeland Wines	X.High	Low	Wine	12.8	White

If two tables have a column of the same name, then they can be joined naturally without specifying which columns to join explicitly.

2.6 Distinct δ

The δ operator will ensure that no rows are duplicated.

2.7 Chaining operators

Just like in normal algebra, you can chain operators:

$\delta(\pi_{Strength,Colour}(\textit{Alcohol-Selection}))$

Gives:

Strength	Colour
11	Red
4.2	Yellow
12.8	White
18	Carmine