Information Modeling

David Edmond

Information Modeling

Javvin Technologies Inc. Distribution

Information Modeling

David Edmond

Javvin Technologies Inc. Distribution

http://www.javvin.com
http://www.networkdictionary.com

Table of Contents

Table of Figures

Chapter 1
Introduction

1.1 Why Compute?

What *are* computers for? What is their purpose? Suppose your life depended upon coming up with a word or phrase that most accurately summed up what computing is all about. What would your answer be?

Would you say that computing is about . . .

sex?
drugs?
rock'n'roll?

No, there's not too much of that in computing.

Well then, perhaps it's about . . .

money?
power?
food?
gambling?

No, these topics are hardly ever discussed in computing magazines.

Chapter 2
Specific Facts

2.1 Introduction

Computers are *not* magical. They are *marvellous*, but they are not magical. They may be extremely fast, with computation speeds measured in millions of instructions per second. They may have huge amounts of memory, measured in billions of characters. But there is nothing happening inside them that we could not contemplate doing ourselves. We may take a lot longer; we may get bored and make mistakes, but we *must* believe that we could. We must think of the computer as doing things that we could do with pencil and paper or with a blackboard and some chalk. If we cannot do this, then we are resigned to thinking of the computer as something beyond our comprehension. As a consequence of this necessary act of faith, it is the things that *we* can express (in conversation with a friend or on a piece of paper, say) that are of importance. And, unless we are day-dreaming, these expressions have some meaning. They are attempting to say something about reality. The **sentence** is the unit of language that allows us to say things about the world in which we live. Sentences, however, come in all shapes and sizes; there are commands, questions, forecasts and opinions to name just a few. This book will focus on one particular category consisting of what are called declarative sentences or, more simply, **facts**. A declarative sentence is one that is capable of being true or false. Consider the following sentences:

Stop, in the name of love!
Big girls don't cry.
Will you still love me tomorrow?

Only one of these three is some kind of statement about the world. Only one is a representation. Only one can be added to the end of:

I declare that:

Chapter 3
Sets

3.1 Introduction

Suppose someone writes down a list of people's names and hands that list to you. Then you are asked what these people have in common.

It is fairly safe to claim that one way or another you would find something to connect these people. Even if the names were as unlikely as John, Paul, George and Ringo. You would probably feel frustrated and disappointed with yourself if you were unable to discern some common feature.

A **set** is a collection of objects, with the objects usually sharing some property. The formation of a set allows us, mentally, to gather things that seem to belong together, and to provide them with a collective being. This process of **generalization** is a means of conquering complexity. Defining a set is a way of enforcing order upon our world and because of that order we can have reasonable expectations. We anticipate certain kinds of behavior and not others.

By isolating an object and stating that this thing is a "man", for example, we accomplish two things:

1. We provide a number of properties that can be ascribed to that object – beards, beer and baldness perhaps.
2. We group this person with other men – all the people who share these properties.

Having decided that a person is a man or a woman or a singer or a computer programmer we would expect a whole range of associated behavior patterns.

3.2 Sets and Everyday Language

There are two ways to specify a set: **set extension** and **set comprehension**. These two methods form an essential part of our everyday language.

Chapter 4
Relations

4.1 Introduction

In this chapter we take a step towards the implementation of our specific facts. In previous chapters, we attempted to represent situations in reasonably natural, if formal, way. We would usually consider a person's age and a person's father to be separate facts about that person; and so, in our specification, we would probably want to treat them separately. Don't forget that the specification is a description written for *our* benefit. An implementation, however, is a description written with automation in mind. While a specification may be written with a relatively free hand, an implementation is usually required to be efficient and effective, using a minimum amount of storage space and providing an acceptable response time.

This chapter provides a continuation of the formal notions of relations and sets that were introduced in the two previous chapters. It allows us to gather these ideas in a theoretical manner before discussing their implementation in a "real-live" computer language, namely SQL.

The chapter introduces the **relational model** of data. Using this approach, facts are combined to produce larger storage structures called relations. A relational database is a cohesive collection of relations. We use the relational model (1) because it allows us to access and to manipulate facts in a relatively easy manner, and (2) because there are many commercially available database management systems that support the relational model.

4.2 Merging Facts

The idea of a relation was introduced in Chapter 2 where it was described as a set of pairs. In that chapter, relations were frequently shown in the form of a two-column table. For that reason, we might call them **binary** relations to distinguish them from the more general relations that are the subject of this chapter. But, because each binary relation corresponds to a particular type of fact, we will also refer to binary relations as **fact types** to make their origin clear. Here are two examples of these fact types that were introduced in that chapter.

Chapter 5
Introducing SQL

5.1 Introduction

In this chapter we introduce one of the most important computer languages so far developed, SQL. It represents a major departure from the languages we usually think of in connection with computer programming. These more conventional languages are primarily concerned with giving instructions to a computer. SQL is different.

SQL is, first and foremost, a **means of communication**, a means of expressing our requirements. These requirements are passed to a complex software product known as a database management system (DBMS). This software is designed to control access to and usage of the database. SQL is a means of telling the DBMS what we want done. Because the nature of the language allows us to concentrate on specifying the information to be retrieved from our database, there is a consequential load placed upon the DBMS. It must be able to determine a sufficiently rapid means of accessing the data, sufficiently rapid, that is, to satisfy our need for the data.

SQL is an acronym for **S**tructured **Q**uery **L**anguage, and the key word is **query**. This word is to be taken in a more general sense than simply “retrieval". The central idea in SQL is that of identifying the portion of the database that interests you. Having done that, you may apply some operation to that portion: you may display it, you may update it, or you may delete it.

This chapter is intended to provide a brief look at some of the language’s major features. These features are divided into four groups concerned with:

- **database definition**, whereby the major components of the database may be defined, modified or discarded;
- **database retrieval**, whereby the portion of the database that meets certain conditions may be identified and examined;
- **database manipulation**, whereby some part of the database may be extended, updated or deleted;

Chapter 6
SQL Retrieval

6.1 Introduction

This chapter contains a series of examples of database retrieval using SQL. The examples attempt to show the basic retrieval capabilities of the language.

There are three basic ways in which information may be extracted or derived from a table. These relate to the ways that we ourselves might extract information presented to us in tabular form.

Sometimes we are interested in detailed information. We scan down particular columns looking for values that interest us, stopping when we find such a value. Then we will examine the rest of the row upon which we found the value. This is how people look up telephone numbers or exam results or sports results or a timetable. The search operation will be repeated until we have, for example, noted our own exam results and those of our friends.

There is another kind of retrieval. This kind is performed when, essentially, we are looking for one particular value. The value may be one that can be extracted from the table, or it may be a derived value. The situations when we scan a table in this way are, for example, when looking for the lowest mark in an exam or the total number of people who passed or the time of the last train or bus.

The third kind of retrieval is the kind performed when we want to compare one group of figures with another. Did chemistry students perform better than computing students? Are there more trains to town than buses?

These are the basic means of retrieval offered by SQL. There is nothing performed by SQL that we could not contemplate doing ourselves. SQL is a language, after all; it is a means of expressing our wishes.

All examples are based on the SUBJECT database introduced in Chapter 4. This database contains three tables:

- `Students`, which contains the names of students enrolled in the single subject offered;
- `Assess`, which contains details of assessment involved in the subject; and

Chapter 7
SQL Modularization

7.1 Introduction

In computing, a **module** is the name we give to an item of work. A program module is a discrete component of that program. It performs a particular task, such as finding the minimum of a set of numbers. All the various modules of a large program are put together in such a way as to achieve the program's overall goal.

This process of conquering complexity is sometimes called **modularization**. Using this technique a complex task may be reduced to a number of relatively simpler tasks. Suitable program modules are then built to accomplish each of these tasks.

This chapter is concerned with how the fundamental query building methods of Chapter 6 may be combined in different ways. In creating these more complex queries we can answer more complex questions.

Three mechanisms are discussed. They are:

- query **nesting** whereby the results of one query are fed into another;
- the **union** operator which allows the results of two or more queries to be merged to produce a single result table; and
- the **view** which allows the results of a query to be given a name and subsequently treated as just another database table.

7.2 Query Nesting

The first kind of query modularization considered in this chapter is query nesting. This involves passing results from one query directly into another. Thus two queries may be executed, one after the other, with no "manual" intervention required.

Example 7.1 What is the Id of the student who got the lowest mark in the first assignment? We could issue the following command:

Chapter 8
Facts and Relations

8.1 Introduction

A **fact** is a declarative sentence; that is, it is a statement which may be either true or false. It describes a particular relationship between two or more things or **entities**; for example:

```
Billy Connolly was born in Scotland.
```

We use computers when we have lots of similar facts to remember.

```
Bill Cosby was born in the USA.
John Cleese was born in England.
Barry Humphries was born in Australia.
 :     :         :   :    :   :
```

When we recognize that certain facts are similar, we can generalize them into a **fact type** which is a relationship between two types of entity rather than between individual entities. In this case the relationship is between people and countries. Or is it? Perhaps it's between comedians and countries, or between men and countries? To be more certain we need to investigate the *universe of discourse* or UoD which is simply the situation that we intend representing in our information system. In attempting to design the most appropriate database structure for a given situation, we need to know the kinds of things that are involved and the kinds of facts that relate them.

Fact types are not stored individually; rather, they are embedded within **relations** with each relation dedicated to representing a fixed number of fact types. This chapter looks at some of the problems we face when deciding where to place a fact type when designing a database. We will find that each fact type may be merged or grouped only with certain others. Some we will be unable to merge; they must remain on their own. We will use **conceptual schema diagrams** to help achieve this grouping. These diagrams are used to depict the knowledge we need to design a database.

Chapter 9
Uncovering Facts

9.1 Introduction

Suppose we are required to design a database to support a new information system. In the preceding chapter some rules were formulated regarding which facts may and which may not be merged into relations. Once we have, in front of us, the kinds of facts that are to be stored in the database then it is a relatively mechanical process to follow these rules and to arrive at a design for the database.

Unfortunately, this information is rarely presented to us in a neatly packaged and labeled way. In other words, the basic facts types do not usually show themselves clearly and obviously. We, the designers, must identify them.

The people who are going to use this new system will want the computer to extract information from the database, to sort it, to merge it with other information, to summarize it, and so on. They are most unlikely to be interested in receiving long lists of quite trivial facts. They have sophisticated ideas of how the organization works and may want these ideas reflected in complex reports.

A report is simply a view of the organization. This chapter introduces a language that may be used to describe the structure of such views. From these descriptions, the underlying simple facts may be uncovered.

9.2 Defining Syntax

The **syntax** of a language is a set of rules that govern exactly what may be said in that language. This definition applies as much to programming languages as it does to any other kind.

The language to be presented in this chapter is a special language used to describe the syntax of programming languages. It is the language in which we write the rules of syntax. The syntax of SQL, for example, tells us that "`Select * From Students`" is legal SQL whereas "`From Students Select *`" is not. Syntax is concerned with the superficial

Chapter 10
Fact-based Analysis

10.1 Introduction

This chapter is presented as a worked example in a technique which we will call **fact-based analysis**. This is a way of designing a relational database. In particular, it is concerned with developing a design that guarantees that in any resulting database each fact is stored *just once*. Here are the stages that we follow.

1. Uncover the relevant entity types and the fact types that join them.

   In this step, we apply the techniques of Chapter 9 to find the relevant elementary fact types.

2. Look for any uniqueness constraints involved in each fact type.

   In this step, we apply the question and answer technique of Section 8.6 to decide whether a fact type is a many-to-many, a many-to-one or a one-to-one relationship.

3. Construct record types by merging fact types, where appropriate.

   In this step, we merge fact type according to the rule that permits the merging of single-valued facts about the same kind of thing.

4. Decide which attributes may be null.

   In this step we process each record type in turn, examining each non-key attribute of that record. Those that may contain null values are flagged.

5. Define the database.

   In this step, we provide an outline of the database.

6. Review the design.

   Finally, we should check that the database design is satisfactory. Has any computable or derivable information slipped through into our design? Using SQL, can the major views be reproduced with this design?

Chapter 11
Entity-relationship Modeling

11.1 Introduction

Entity-relationship modeling is a very popular method for designing databases. ER modeling, as it is often called, may be described as a **top-down** approach in that it encourages to look at the “big picture" first. We begin by describing the world in terms of **entity** types that are **related** to one another in various ways. We may then refine that picture to show the **attributes** of each entity type. Thus we start by looking for the major kinds of things that populate the situation to be modeled. These entity types will give rise, eventually, to the major relations in our database. In a hospital situation, for example, the entity types might be:

- patients
- wards
- beds
- surgeons
- nurses

We then establish any relationships that exist between these entity types, such as:

- Patients are operated on by surgeons.
- Patients are located in beds.
- Beds are placed in wards.
- Nurses are allocated to wards.

Chapter 12
Knowledge

12.1 Introduction

This chapter is an introduction to the **Z Notation** which is a language that allows us to express our understanding of any given situation in a concise and precise way. In Chapter 1, it was suggested that if we write down all that we know about something, then our statements may be divided into (1) simple specific statements that were termed **facts** and (2) more general statements that were termed **knowledge**. This chapter is concerned with the more general statements, that is, it is about knowledge representation. Z is the language we will use to express that knowledge.

The generality of knowledge is achieved by the statements involved saying something about whole classes or sets of objects. Such knowledge will eventually be encoded as computer programs, so Z will be used to **specify** these programs. The resulting specification will then be **implemented** using a programming language. It is important, however, to realize that the implementation is yet another description of the *same* situation that was portrayed in the original specification. The implementation is a kind of re-specification of that situation, this time written in a specification language that the computer can follow and obey or execute. This implementation is, in effect, an **executable** specification; that is, it will be written using a programming language such as C or COBOL or SQL or some combination of these.

The Z Notation is a particular style of writing two mathematical languages, **set theory** and **predicate calculus**. Z is an amalgam of these. Most of the ideas relating to set construction and manipulation were discussed in Chapter 3. In this chapter, we will see how to write set expressions in Z. Mostly, this involves using a special symbol rather than a word. For example, Z uses the symbol $\cup$ for set union whereas in Chapter 3 and in the coverage of SQL that followed, we would have used the word `union` instead.

The second element of Z is predicate calculus, and many of the ideas behind this theory have also been discussed previously. This was done in Chapter 2 where we discussed basic simple facts and their construction. In Z, basic facts are also constructed in the same way,

Chapter 13
The Knowledge Base

13.1 Introduction

Documentation is one of the most disliked features of computing. This is rather unfortunate since program documentation is where we store our knowledge in its most (human) readable form. As an example, consider a simple rule stating that a customer's current balance must not be allowed to exceed his or her credit limit. A rule like this is typically specified using a program design technique such as pseudocode or a decision table. The rule might then be encoded within a COBOL program. Thus the program becomes the rule enforcer. The rule is specified in pseudocode and implemented in COBOL.

What happens next? If we are honest with ourselves we know that from now on all attention turns to the program code; the specification takes on a secondary role of documentation. Changes to the rule occur, such as amendments, extensions, special cases and so on. These are implemented directly in the program and the documentation becomes increasingly obsolete, simply confirming the programmer's prejudice against documentation. The result is that the database is encapsulated by a collection of programs, with each program implementing any number of undocumented rules.The database starts to suffer from hardening of its arteries. Organizational knowledge becomes buried in programs. When the system is to be replaced, all this knowledge must be rediscovered by the next generation of systems developers.

One answer to this problem is to keep the documentation up to date and to make the programs subordinate to this documentation. In doing so, we would move towards a more evolutionary style of systems development. Maybe the term *specification* should be discarded. In the minds of many programmers, any specification is a disposable means to an end; the end being an executable program. A better approach is to consider that the programs constitute a knowledge-base of some kind; and that knowledge needs to be expressed in at least two forms: (1) in a way that humans can understand; and (2) in a way that machines can execute. Both forms are necessary; the first for us, and the second for the machines. A knowledge base of the kind being proposed is simply continuously updated

Chapter 14
From Specification to Implementation

14.1 Introduction

In this chapter we will look at how the various schemas that form a Z specification might be put together. The situation to be modeled is that of a class of students who are studying a particular subject. The specification covers such typical activities as students being enrolled, being awarded marks, having marks amended and, hard to believe, students dropping the subject. We will begin by introducing a **state schema** which provides a static picture of the classroom. Based on that picture we specify a number of **operation schemas** which describe the ways in which the classroom may change. Then we return to the state schema and discuss how it might be developed. Finally, we discuss the relationship between this **specification**, which consists of a state schema and a number of operation schemas, and the **implementation**, which consists of a database and a number of programs.

14.2 The State Schema

Suppose we describe the classroom in the following way.

Class

$$
\begin{array}{l}
students : Set\ of\ Person \\
last : Person \pfun Name \\
mark : Person \pfun 0..100 \\
\hline
dom\ last = students \\
dom\ mark \subseteq students
\end{array}
$$

Chapter 15

Database Definition in SQL

15.1 Introduction

This chapter describes how to define the major objects that may appear in an SQL database.

- There are **tables** without which the database would be empty. These are sometimes referred to as **base** tables.
- There are **views** which define **virtual** tables.
- There are **indexes** which enable the DBMS to respond to queries within an acceptable period of time.

The word definition is used in the general sense of describing or delimiting the properties of these objects. So, this chapter will discuss the creation, alteration and, in the extreme case, removal of these properties.

Information about the properties of database objects is stored in the **system catalog** or dictionary. The catalog itself takes the form of a set of tables which we may examine ourselves using SQL. It forms a relational database that resides alongside our own.

15.2 Tables

15.2.1 Table Creation

A new table is introduced into the database by means of the `create table` statement which, in its simplest form, only requires that we name the table and then name and provide a datatype for each column in the table.

Chapter 16
Database Manipulation in SQL

16.1 Introduction

In this chapter we look at the three data manipulation commands of SQL.

- The `insert` statement enables one or more new rows to be added to a table.
- The `update` statement allows one or more existing rows to be modified in some way.
- The `delete` statement allows one or more existing rows to be removed from a table.

These statements allow the user's model (or information system) to be *manipulated* so as to reflect changes that have occurred in the user's environment.

16.2 Adding New Rows

The `insert` statement allows one or more rows to be inserted into a table.

There are two forms of the statement, one that allows us to insert a single row, and one that allows the insertion of a set of rows.

16.2.1 Single Row Insert

This form allows one new row to be placed in the table specified.

Example 16.1 We want to add a new student number 999, with the name Meg Murphy. To do this we construct the row with a `values` clause and then insert that into the `Students` table.

```
Insert
Into Students
Values(999, 'Meg','Murphy')
```

Chapter 17
Application Programming

17.1 Introduction

This chapter is concerned with how we program our information systems or **application**. The system will, typically, consist of a database and a set of programs. This split reflects that division first discussed in Chapter 1.

- The database contains simple specific facts concerning the organisation.
- The programs contain more general statements or knowledge.

The programs themselves can be divided into two groups according to the kind of knowledge they encode. Some of them are **report** programs. They inspect the database, make calculations using the data retrieved, and report on the results of their calculations. The other group of programs **process transactions**, that is, they allow events and changes in the real world to be represented in the database. Each of these latter programs will be dedicated to handling one particular kind of event.

A large part of the work involved in operating an organisational information system is concerned with this latter group of programs. The information system *must* be able to record changes in the organisation's environment and circumstances.

This chapter is mainly concerned with such transaction processing. It examines how the operation schemas that are used to *specify* the transactions are *implemented* as SQL programs.

Each operation schema could be divided into two separate sets of conditions.

- There are the pre-conditions which, collectively, state what conditions must apply before some event may truly be said to have occurred.
- There are the post-conditions which, on the basis of the pre-conditions being satisfied, say how the situation changes as a result of this event.

Chapter 18
Case Studies

18.1 Introduction

In this chapter we will look at two data processing situations.

- The first situation involves deriving a compact yet complex report or view from some simple data.
- The second situation involves monitoring the handling of orders made on a small manufacturer.

For both situations, we will (1) present an informal introduction, (2) describe it formally, and (3) look at an implementation in SQL.

18.2 The League Table

18.2.1 Introduction

Last year, a number of rugby clubs agreed to take part in a competition to decide the best team in the district. The competition is to take place over a number of weeks. Every week there will be a round of matches with each team playing one match per round. There are six clubs altogether, so each round will involve three games. In the first round, the results were as follows:

Round 1 Results

Home Team		Away Team	
Wiseacres	12	Shinhackers	8
Rosewell	8	Witsend	20
Rovers	5	Jeeps	5

The convention about presenting results is that the home team and its score are given first and the away team and its score second. Each team will play all the others twice, once at

Chapter 19
Refinement

19.1 Introduction

This chapter is about how we implement our specification, that is, it is about how we turn it into a collection of computer programs operating upon a database. The situation we want to reach is one where we will have two quite distinct pictures *of the same situation*.

We start with one picture, the one provided by the specification. This will be stated or expressed in a language that tries to describe the situation as *we* see it. The other picture is a re-statement of that same situation; but this time the language used is ambivalent. It can be taken as just another way of perceiving the problem, but it can also be thought of as providing instructions to a machine in order for that machine to create an animated equivalent of the original specification. It is a version that is *executable* by the machine. In other words we have made the original problem tractable to information technology.

This use of different forms of language is not restricted to computing. We would give a stranger to town instructions expressed differently from those given to an obvious local asking directions. We would talk to the local in terms of shared knowledge such as familiar streets and landmarks. Conversely, we would talk to the stranger in physical terms – “turn left", “straight ahead for 2 km", “third on the left" and so on.

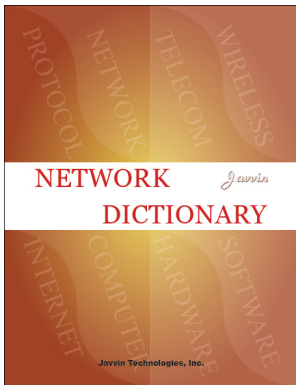
In this chapter we will take a relatively abstract specification such as shown in Chapter 14 and show how to map that to another specification this time expressed in the relational calculus or tuple oriented set comprehension of Chapter 4. This language is the basis for SQL and it will be assumed that the transformation to SQL is straightforward.

This chapter provides a worked example of how to move, formally, from a specification to its implementation. The technique used is *data refinement*, and here it is used on a database system.

A small situation is described along with some of the events that might impinge upon it. The description or specification is written using the Z notation. The intention is to explain the situation as the user sees it.

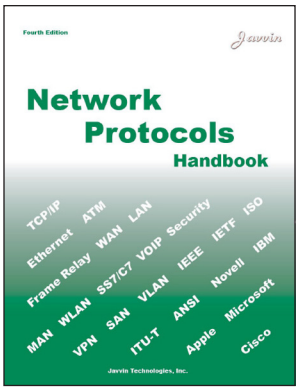
That same situation is then recast in terms of tables and the events in terms of operations

Javvin Networking Technology Series



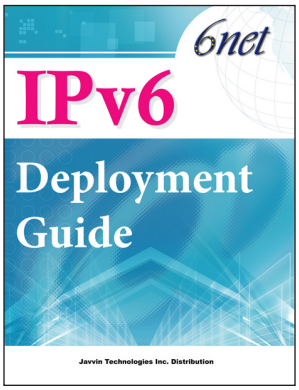
Network Dictionary

Networking, Internet, telecom, wireless, computer, hardware and software - multiple dictionaries in one. A “Must have” reference for IT/Networking professionals and students!



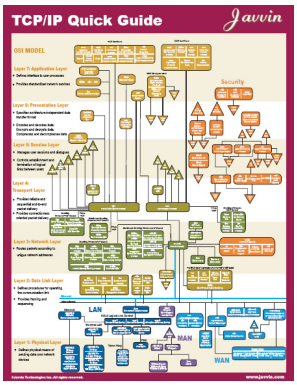
Network Protocols Handbook

Fully explains and reviews all active protocols. Illustrates latest networking technologies.



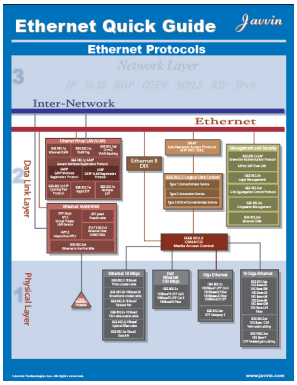
IPv6 Deployment Guide

IPv6 is replacing IPv4 to dominate the networking world. This deployment guide will enable you to fully harness the power of IPv6. A “Must have” reference for IT/Networking professionals and students!



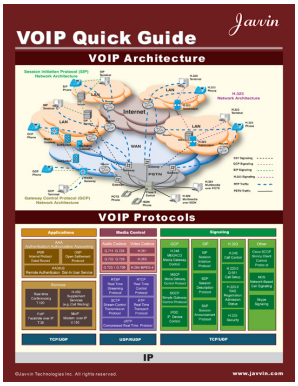
TCP/IP Quick Guide

Practical TCP/IP information extracted from hundreds of pages of TCP/IP books. A comprehensive and clear map of all TCP/IP protocols in OSI 7 layers model. A portable tool for you to carry, insert into a folder or put on your desk.



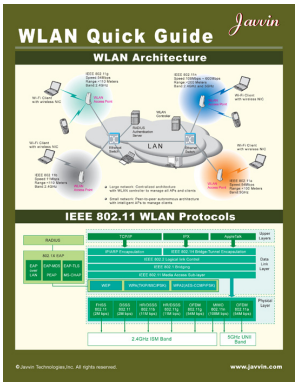
Ethernet Quick Guide

Practical Ethernet information extracted from hundreds of pages of Ethernet books. A comprehensive and clear map of all Ethernet protocols. A portable tool for you to carry, insert into a folder or put on your desk.



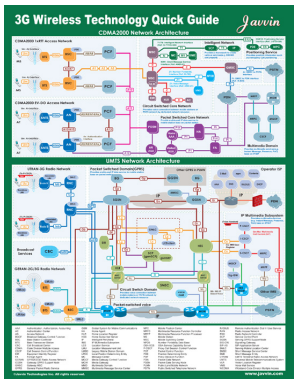
VOIP Quick Guide

All must known VOIP technologies included in this comprehensive yet portable quick reference.



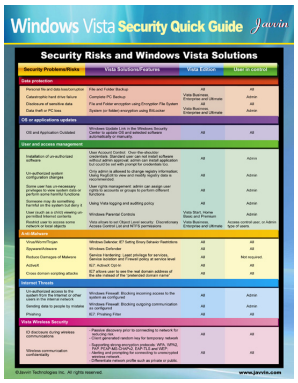
WLAN (WiFi) Quick Guide

A comprehensive quick reference to assist you in WiFi WLAN implementation, learning and operation.



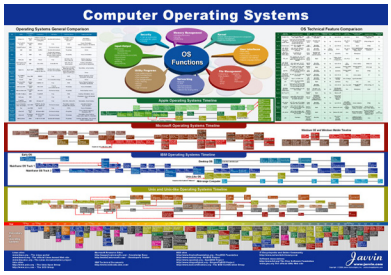
3G Wireless Tech Quick Guide

Highlights the third generation wireless technologies in one portable quick guide.



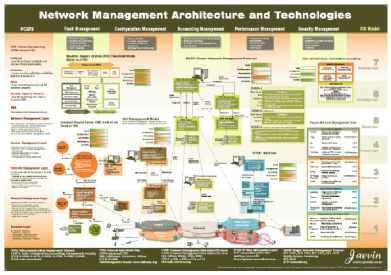
Windows Vista Security Quick Guide

All you must to know about Windows Vista Security on this handy quick guide...useful for all Windows Vista users!



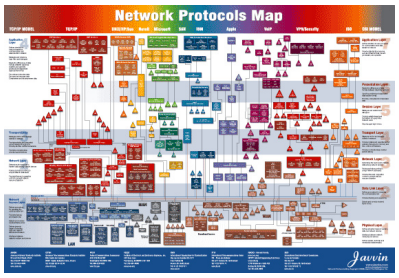
Computer Operating Systems (OS) Poster

Illustrates the computer operating systems now and their evolution path over the past 60 years.



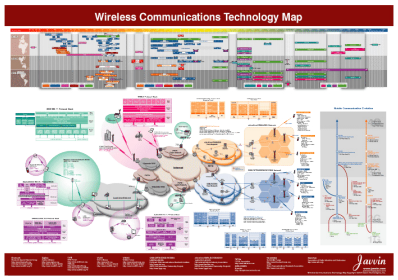
Network Management Architecture and Technology Map

All network management architecture and technologies for both telecom and data communications displayed on one chart.



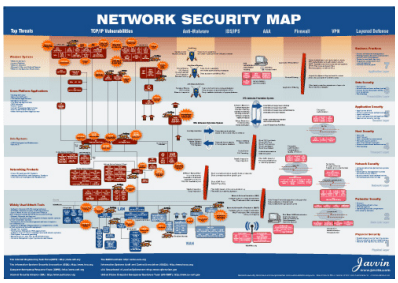
The Network Protocols Map Poster

All network protocols illustrated on one Chart. A "must have" for all networking, IT and Telecom professionals and students.



Wireless Communications Technology Map Poster

All major wireless technologies displayed in one chart: WLAN, WiMAX (WMAN), WPAN and mobile wireless technologies (WWAN)...



Network Security Map

All you must know about network security on one chart! A unique gift for yourself, your colleagues, partners and customers.

www.NetworkDictionary.com

Free networking technology library, comprehensive network protocol and network security knowledge, telecom encyclopedia, computer hardware and software terms and white-papers.

A site for you to learn and share knowledge, ask experts, write blogs and get connected with peers.

Information Modeling

ISBN: 978-1-60267-015-0