Chapter 01
Introduction

In this chapter, we introduce good information that is derived from raw facts. These raw facts are known as data. Data are likely to be managed most efficiently when they are stored in a database. You will learn what a database is, and what it does. You will also learn about various types of databases and why database design is so important as well. File system data management is now largely understood as the characteristics of stored file systems. Database and database management systems have become essential for managing businesses, governments, schools, universities, banks, etc. Also in the chapter, we will talk about the responsibility of a database administrator, and the architecture of 2-tier and 3-tier.

In this chapter, you will learn:

1.1 Database and Database Systems

Data is the keyword to describe the systematic storage of data. Database Management System (DBMS) is the collection of interrelated and persistent data and is a set of application programs used to access, update and manage.

Since the computer was developed, it has been used in various fields as follows.

- computing machine = H/W + numerical analysis S/W
- database = H/W + database management system S/W
- automatic processing system = H/W + process control S/W
- typing machine = H/W + word processor
- game station = H/W + game S/W
- audio/video station= H/W + audio /video playing S/W
- telephone = H/W + voice networking S/W
- etc.

A database is systematic storage of data and a database management system is a system software to build a computerized database.

A database management system consists of

- A collection of interrelated and persistent data is usually referred to as a database (DB).
- A set of application programs used to access, update and manage that data from the database management system (DBMS).

The goal of a DBMS is to provide an environment that is both convenient and efficient to use in

- Storing information into the database.
- Retrieving information from the database.

Databases are usually designed to manage large bodies of information as below:

- Definition of structures for information storage (data modeling).
- Provision of mechanisms for the manipulation of information (file and systems structure, query processing).
- Providing for the safety of information in the database (crash recovery and security).
- Concurrency control if the system is shared by users.

1.2 History

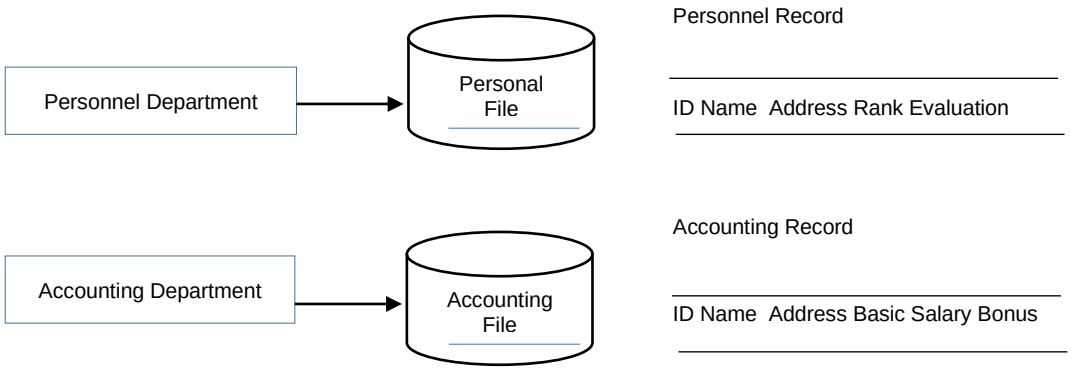
File Systems were types of data used to store, retrieve and update. The Data were stored as a file system with complexity difficult to control then data models were developed. Those models are the hierarchical model and the network model. The size of data usage increased rapidly and also those models are difficult to use while relational databases have appeared.

1.2.1 File System

Databases have been historically an essential part of any information processing system. The origins of databases go back to libraries, governmental, business, and medical records. There is a very long history of information storage, indexing, and retrieval.

Before the 1960s, file systems were used to store, retrieve and update data. The structure is shown in Figure 1.1, and each department manages its files respectively.

Figure 1.1 File System



To see why database management systems are necessary, let's look at a typical "file-processing system" supported by a conventional operating system. The application is a savings bank:

- Savings account and customer records are kept in permanent system files.
- Application programs are written to manipulate files to perform the following tasks:
  - Debit or credit an account.
  - Add a new account.
  - Find an account balance.
  - Generate monthly statements.

Development of the system proceeds as follows:

- New application programs must be written as the need arises.
- New permanent files are created as required.
- But over a long period files may be in different formats.
- Application programs may be in different languages.

So we can see there are problems with the straight file-processing approach:

- Data redundancy and inconsistency
  - Same information may be duplicated in several places.
  - All copies may not be updated properly.
- Difficulty in accessing data
  - May have to write a new application program to satisfy an unusual request.
  - e.g. find all customers with the same postal code.
  - Could generate this data manually, but a long job...
- Data isolation
  - Data in different files.
  - Data in different formats.
  - Difficult to write new application programs.
- Multiple users
  - Want concurrency for faster response time.
  - Need protection for concurrent updates.
  - E.g. two customers withdrawing funds from the same account at the same time - account has $500 in it, and they withdraw $100 and $50. The result could be $350, $400 or $450 if no protection.
- Security problems
  - Every user of the system should be able to access only the data they are permitted to see.

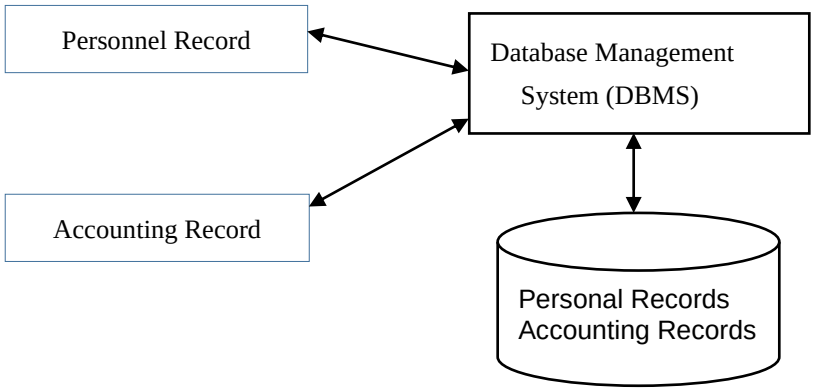
  - e.g. payroll people only handle employee records, and cannot see customer accounts; tellers only access account data and cannot see payroll data.
  - Difficult to enforce this with application programs.
- Integrity problems
  - Data may be required to satisfy constraints.
  - e.g. no account balance below $25.00.
  - Again, difficult to enforce or change constraints with the file-processing approach

These problems and others led to the development of database management systems.

1.2.2 Network and Hierarchical Databases

In the 1960s, computers become cost-effective for private companies along with the increased storage capacity of computers. Databases (DB) and Database Management Systems (DBMS) were developed. Figure 1.2 shows DB and DBMS.

Figure 1.2 DB and DBMS



Two main data models were developed: the hierarchical model (IMS) and the network model (CODASYL) in which access to the database is through low-level pointer operations linking records. Storage details depended on the type of data to be stored. Thus adding an extra field to the database requires rewriting the underlying access/modification scheme. Figure 1.3 shows the hierarchical model and Figure 1.4 shows the network model.

Figure 1.3 Hierarchical model

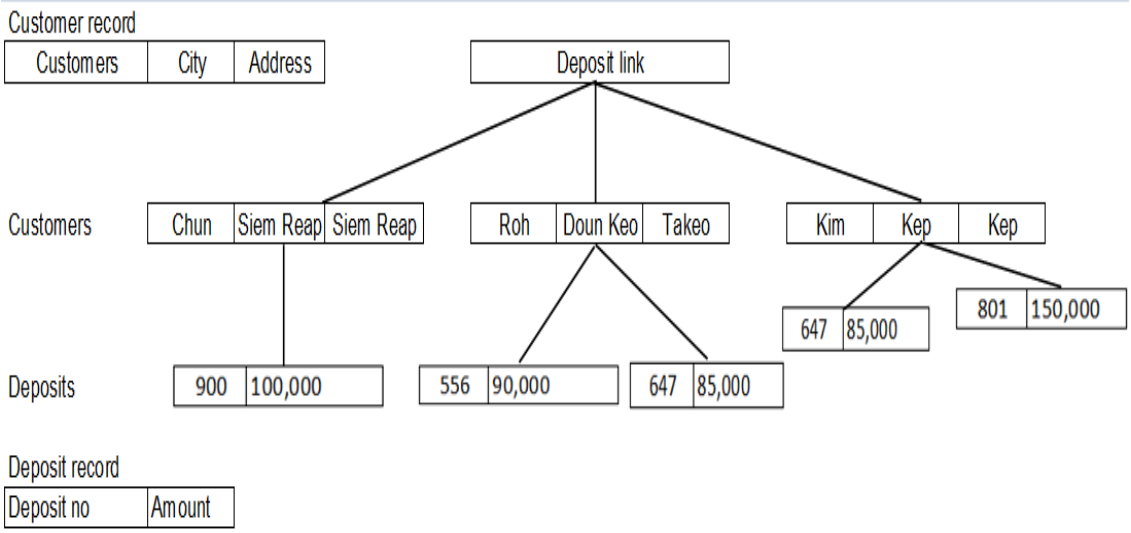
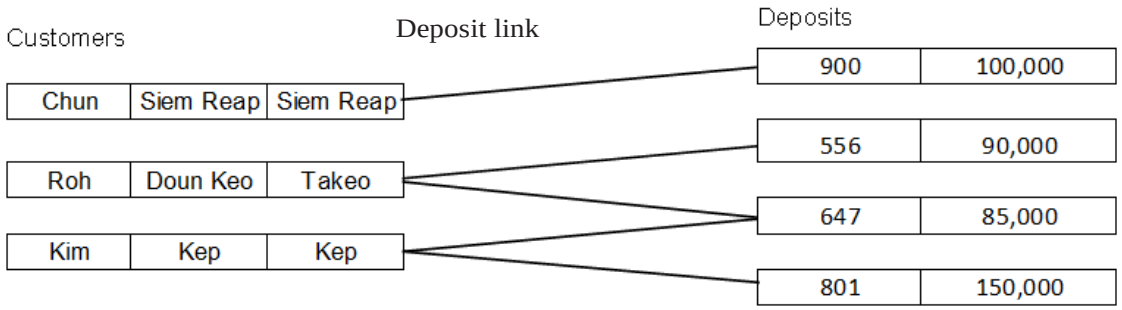


Figure 1.4 Network model



The emphasis in the above two models was on records to be processed, not the overall structure of the system. A user would need to know the physical structure of the database in order to query for information. One major commercial success was the SABRE system from IBM and American Airlines.

In the 1970s, several camps of proponents argue about the merits of these competing systems while the theory of databases leads to mainstream research projects. Two main prototypes for relational systems were developed during 1974-77.

1.2.3 Relational Database

In 1970-72, E.F. Codd at IBM proposed a relational model for databases in a landmark paper on how to think about databases. He disconnects the schema (logical organization)

of a database from the physical storage methods. This system has been standard ever since. In the relational model, records are stored only in tables shown in Figure 1.5.

- Ingres: Developed· at UCB (University of California, Berkeley). This ultimately led to Ingres Corp., Sybase, MS -SQL Server,. This system used QUEL as query language.
- System R: Developed at IBM (International Business Machines) San Jose and led to IBM's SQL/DS and DB2, Oracle, HP's Allbase, Tandem's Non-Stop SOI-,. This system used SEQUEL as query language.

Figure 1.5 Relational model

Customers

Name	City	Address	Account
Roh	Doun Keo	Takeo	556
Roh	Doun Keo	Takeo	647
Kim	Kep	Kep	801
Kim	Kep	Kep	647
Chun	Siem Reap	Siem Reap	900

Deposits

Account	Amount
900	100,000
556	90,000
647	85,000
801	150,000

These systems provide nice examples of how theory leads to best practice. The term Relational Database Management System (RDBMS) is coined during this period.

In 1976, P. Chen proposed the Entity-Relationship (ER) model for database design giving yet another important insight into conceptual data models. Such higher-level modeling allows the designer to concentrate on the use of data instead of logical table structure.

In the early 1980s, the commercialization of relational systems begins as a boom in computer purchasing fueled the DB market for business. In the mid-1980s, SQL (Structured Query Language) becomes the "intergalactic standard", and DB2 becomes IBM's flagship product.

Network and hierarchical models fade into the background, with essentially no development of these systems today but some legacy systems are still in use. Development of the IBM PC gives rise to many DB companies and products such as RIM, RBASE 5000, PARADOX, OS/2 Database Manager, Dbase III, IV (later Foxbase, even later Visual FoxPro), Watcom SQL.

In the early 1990s, an industry shakeout begins with fewer surviving companies offering increasingly complex products at higher prices. Much development during this period centers on client tools for application development such as PowerBuilder (Sybase), Oracle Developer, VB (Microsoft), etc. The client-server model for computing becomes the norm for future business decisions. Development of personal productivity tools such as Excel/Access (MS) and ODBC. This also marks the beginning of Object Database Management Systems (ODBMS) prototypes.

In the mid-1990s, the usable Internet/WWW appears. A mad scramble ensues to allow remote access to computer systems with legacy data. Client-server frenzy reaches the desktop of average users with little patience for complexity while Web/DB grows exponentially.

1.2.4 Now and the Future

In the late '90s, the large investment in Internet companies fueled tools market boom for Web/Internet/DB connectors. Active Server Pages, Front Page, Java Servlets, JDBC, Enterprise Java Beans, ColdFusion, Dream Weaver, Oracle Developer 2000, etc. are examples of such offerings. Open-source solutions come online with the widespread use of GCC, CGI, Apache, MySQL, etc. Online Transaction processing (OLTP) and online analytic processing (OLAP) comes of age with many merchants using point-of-sale (POS) technology on a daily basis.

In the early 21st century, the decline of the Internet industry as a whole but solid growth of DB applications continues. More interactive applications appear with the use of PDAs, POS transactions, consolidation of vendors, etc. Three main (western) companies predominate in the large DB market: IBM (buys Informix), Microsoft, and Oracle.

Future trends show that huge (terabyte) systems are appearing and will require novel means of handling and analyzing data. Large science databases such as genome projects, geological, national security, and space exploration data. Clickstream analysis is happening now. Data mining, data warehousing, and data marts are commonly used techniques today. More of this in the future without a doubt. Smart/personalized shopping using purchase history, time of day, etc.

Successors to SQL (and perhaps RDBMS) will be emerging in the future. Most attempts to standardize SQL successors have not been successful. SQL92, SQL2, and SQL3 are still underpowered and more extensions are hard to agree upon. Most likely this will be

overtaken by other emerging techniques. XML with Java for databases is the current poster child of the "next great thing". Check-in tomorrow to see what else is news.

Mobile database use is a product now coming to market in various ways. Distributed transaction processing is becoming the norm for business planning in many arenas.

Probably there will be a continuing shakeout in the RDBMS market. Linux with Apache supporting MySQL (or even Oracle) on relatively cheap hardware is a major threat to high-cost legacy systems of Oracle and DB2 so these have begun pre-emptive projects to hold onto their customers.

Object Oriented Everything, including databases, seems to be always on the verge of sweeping everything before it. Object Database Management Group(ODMG) standards are proposed and accepted and maybe something comes from that.

Ethical/security/use issues tend to be diminished at times but always come back. Should you be able to consult a database of the medical records/genetic makeup of a prospective employee? Should you be able to screen a prospective partner/lover for genetic diseases? Should amazon.com keep track of your book purchasing? Should there be a national database of convicted sex offenders/violent criminals /drug traffickers? Who is allowed to do Web tracking? How many times in the last six months did you visit a particular sex chat room/porn site/political satire site? Who should be able to keep or view such data? Who makes these decisions? The history of database research is shown in Figure 1.6.

1.3 DBMS

The evolution of DBMS and a broad spread rapidly to improve data sharing, reduce data redundancy, and are also a program data independence. The responsibility of the database administrator (DBA) and the architecture of 2-tier, 3-tier, and n-tier are described in the point as well.

1.3.1 Structure

A DBMS can be an extremely complex set of software programs that controls the organization, storage, and retrieval of data (fields, records, and files) in a database. The basic functionalities that a DBMS must provide are:

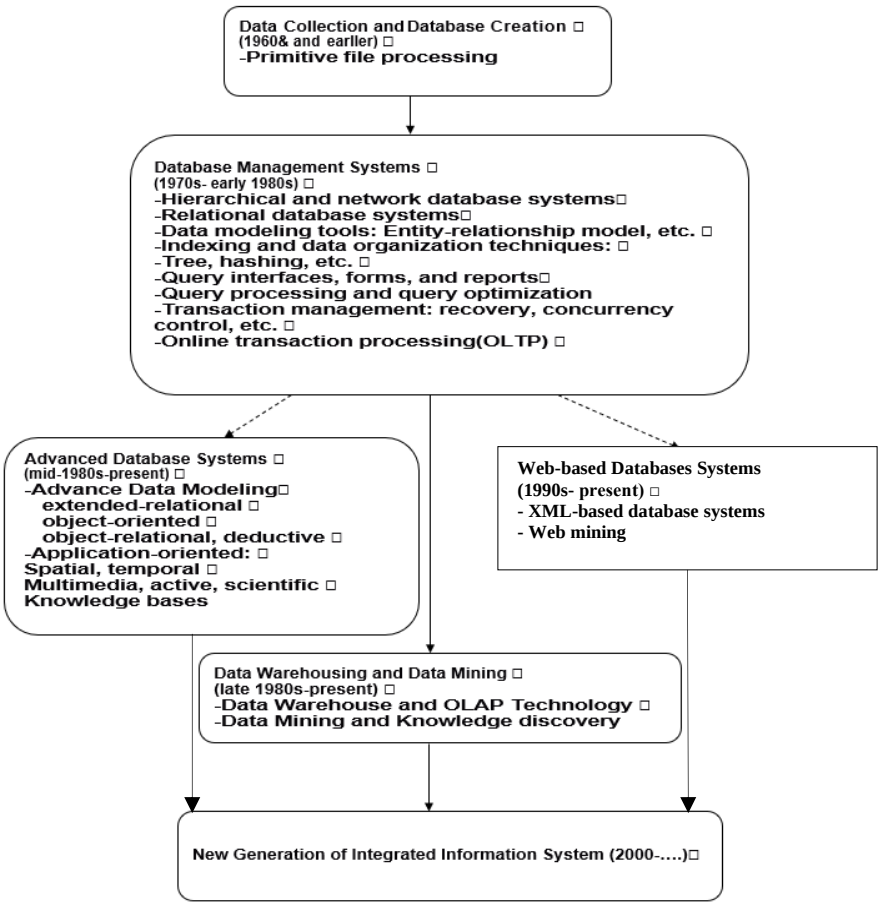
1. A model language to define the schema of each database hosted in the DBMS, according to the DBMS data model.

2. Data structures optimized to deal with big amounts of data recorded to a permanent data storage device, which are very slow compared to the primary storage (volatile main memory).

3. A database query language and report writer to allow users to interactively interrogate the database, analyze its data and update it according to the users' privileges on data.

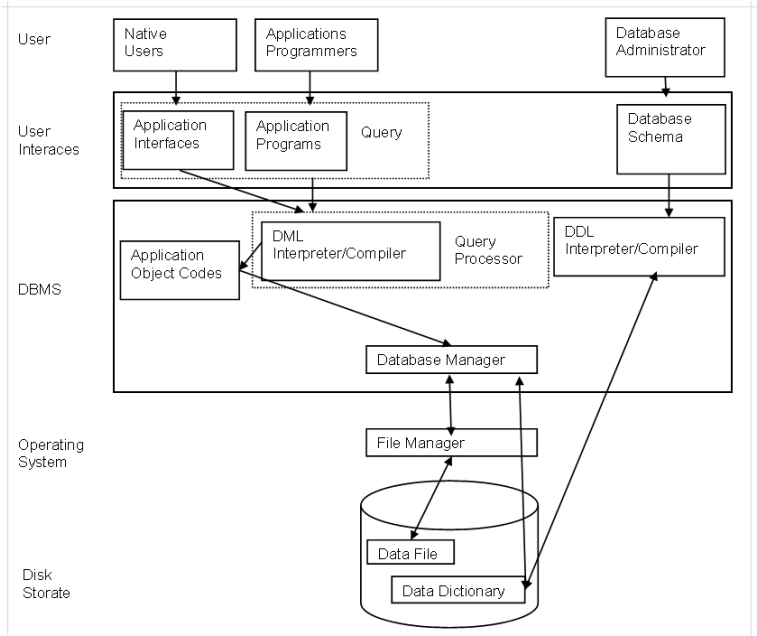
4. A transaction mechanism, that ideally would guarantee the ACID properties, in order to ensure data integrity despite concurrent user access (concurrency control) and faults (fault tolerance).

Figure 1.6 the history of database



The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data. The overall structure of the DBMS is shown in Figure 1.7

Figure 1.7 structure of the DBMS



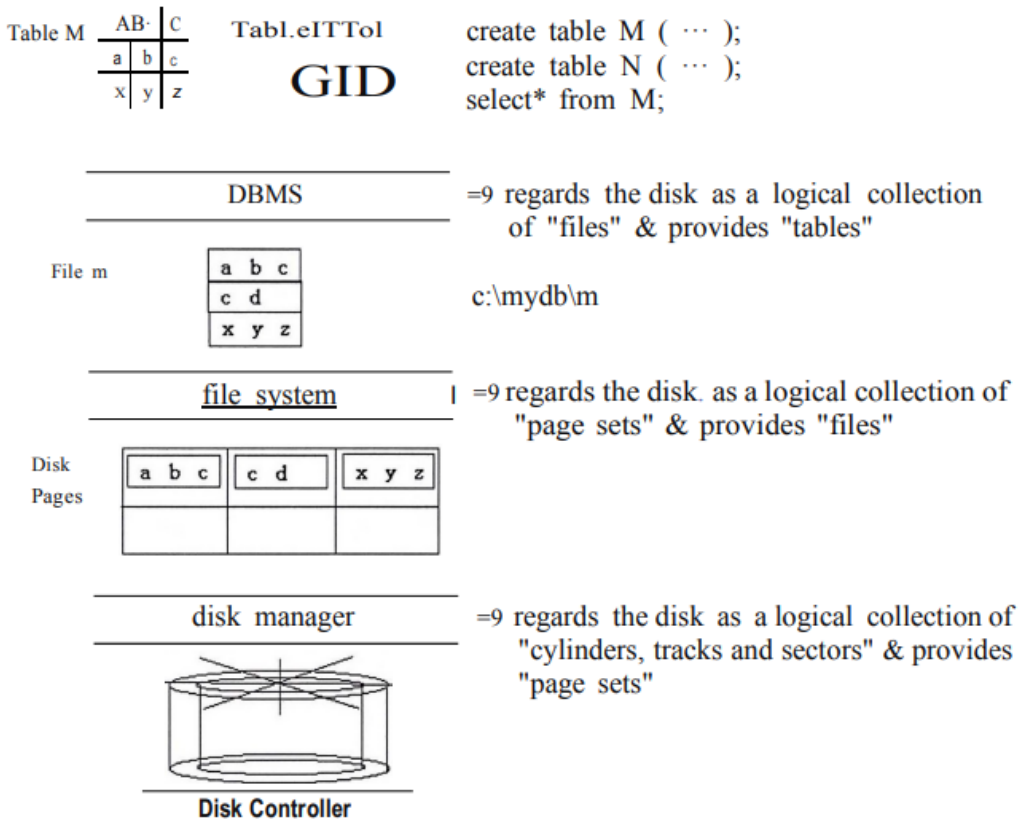
The data structures in the database are as follows.

- data files: store the data.
- indices: provide for fast access to data items.
- data dictionary (= System catalog): store info about the structure of the DB.
- statistical data: store info about the statistical data in the DB.
- log file: store the insertion/deletion/update of the relations for recovery when a system crash happens.

Data abstraction

The DBMS provides users with tables by using layered abstraction shown in Figure 1.8

Figure 1.8 DBMS tables by using layered abstraction



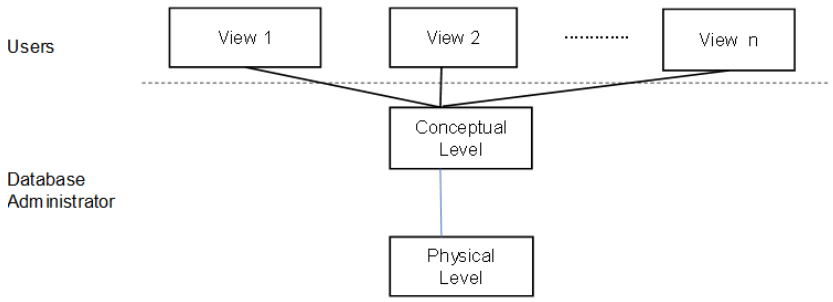
The major purpose of a database system is to provide users with an abstract view of the system. The system hides certain details of how data is stored and created and maintained. Complexity should be hidden from database users.

There are several levels of abstraction from the bottom to the top:

1. Physical Level:
   - Lowest level of abstraction.
   - How the data are stored.
   - e.g. index, B-tree, hashing.
   - Complex low-level structures described in detail.
2. Conceptual Level:
   - Next highest level of abstraction.
   - Describes what data are stored.
   - Describes the relationships among data.
   - Database administrator level.

3. View Level:
   - Highest level.
   - Describes part of the database for a particular group of users.
   - Can be many different views of a database.
   - E.g. tellers in a bank get a view of customer accounts, but not of payroll data.

Figure 1.9 The three levels of data abstraction



When a DBMS is used, information systems can be changed much more easily as the organization's information requirements change. New categories of data can be added to the database without disruption to the existing system.

DBA

A database administrator (DBA) manages the database and the DBMS shown in Figure 1.7. The tasks performed by the DBA are categorized as follows.

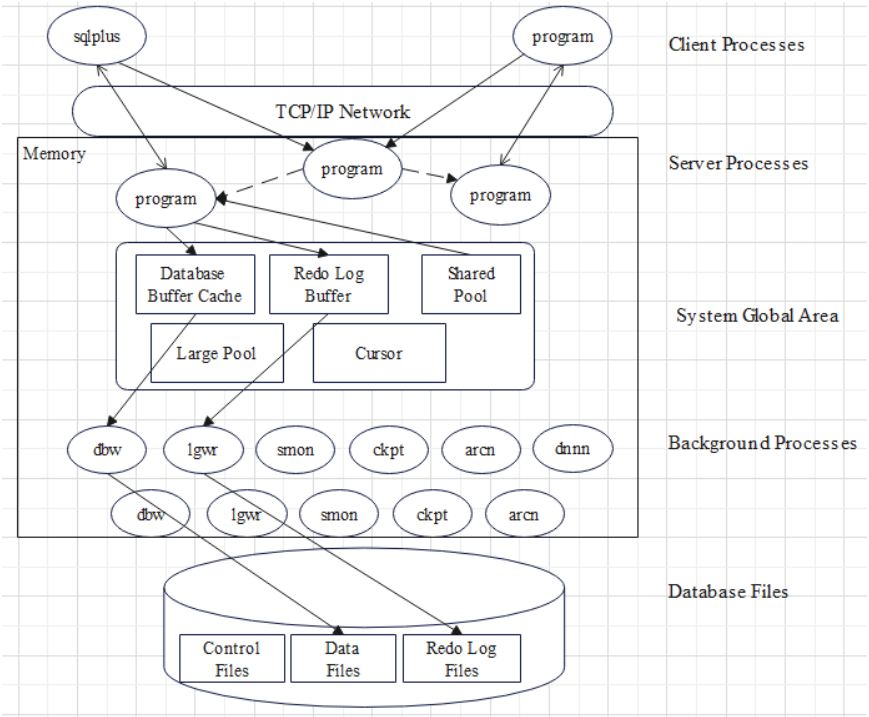
- DBMS management
- database management
- object management: tables /views/indices/clusters etc.
- user management
- performance tuning/backup

Organizations may use one of kind of DBMS for daily transaction processing and then move the detail onto another computer that uses another DBMS better suited for random inquiries and analysis. Overall systems design decisions are performed by data administrators and systems analysts. Detailed database design is performed by data administrators.

Database servers are specially designed computers that hold the actual databases and run only the DBMS and related software. Database servers are usually multiprocessor computers with RAIQ (Redundant Arrays of Independent Disks) disk arrays used for stable storage.

Connected to one or more servers via a high-speed channel, hardware database accelerators are also used in large-volume transaction processing environments. The instance of the Oracle server is shown in Figure 1.10

Figure 1.10 the Oracle DBMS Server



A number of background processes are run in the main memory.

- DBW: Database Writer
- LGWR: Log Writer
- SMON: System Monitor
- CKPT: Checkpoint
- ARCn: Archiver
- Dnnn: Dispatcher
- RECO: Recoverer
- PMON: Process Monitor
- LCK0: Lock

– SNPn: Job Queue
– QMNn: Queue Monitor

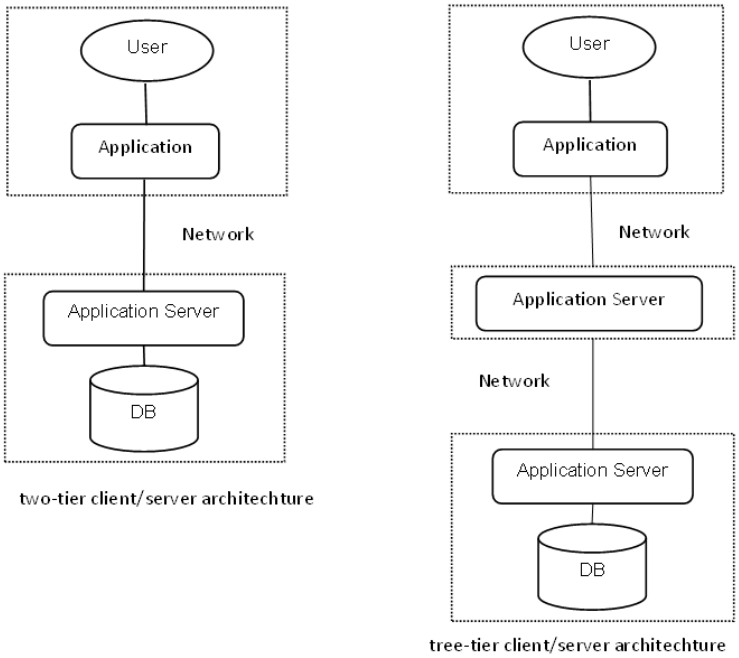
1.3.2 Client/Server and Web

Through the appearance of Local-Area-Networks, PCs came out of their isolation, and were soon not only being connected mutually but also to servers.

Client/Server-computing was born. Servers today are mainly file and database servers; application servers are the exception. However, database servers only offer data on the server; consequently, the application intelligence must be implemented on the PC (client). Since there are only the architecturally tiered data server and client, this is called 2-tier architecture.

This model is actually the opposite of its popular terminal-based predecessor that had its entire intelligence on the host system. The 2-tier and 3-tier client/server models are shown in Figure 1.11.

Figure 1.11 the 2-tier and 3-tier client/server models



One reason why the 2-tier model is so widespread is because of the quality of the tools and middleware that has been most commonly used since the ‘90s: Remote- SQL, ODBC,

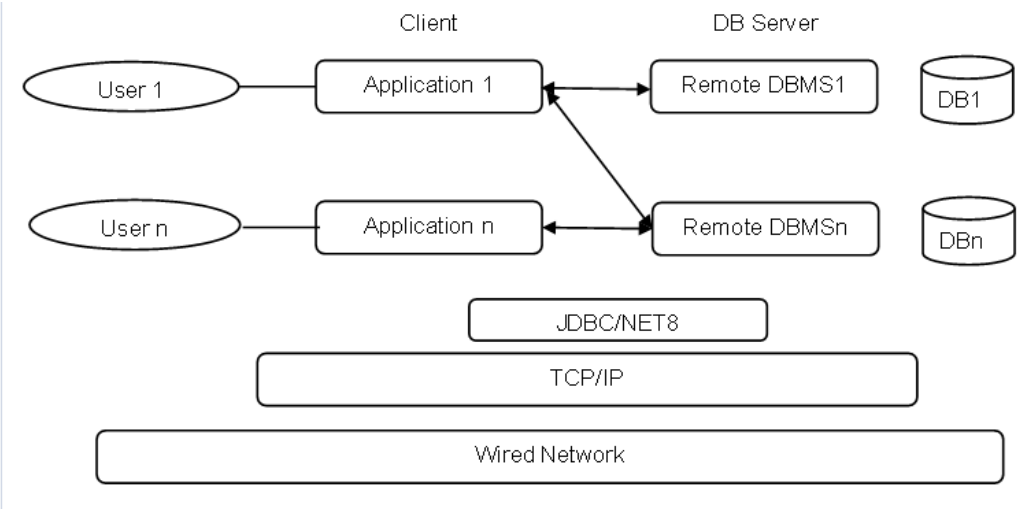
relatively inexpensive and well-integrated PC tools (like Visual Basic, Power-Builder, MS Access, 4-GL-Tools by the DBMS manufactures).

In comparison, the server side uses relatively expensive tools. In addition, the PC-based tools show good Rapid-Application-Development (RAD) qualities i.e. those simpler applications can be produced in a comparatively short time. The 2-tier model is the logical consequence of the RAD tools' popularity: for many managers, it was simpler to attempt to achieve efficiency in software development using tools, than to choose the steep and stony path of "brainware".

Why 3-tier?

Unfortunately, the 2-tier model shows striking weaknesses that make the development and maintenance of such applications much more expensive. The software connection of the 2-tier model is shown in Figure 1.12, and the problems are listed below.

Figure 1.12 the software connection in the 2-tier model



- The complete development accumulates on the PC. The PC processes and presents information which leads to monolithic applications that are expensive to maintain. That's why it's called a "fat client".
- In the 2-tier architecture, business. -logic is implemented on the PC. Even though the business logic never makes direct use of the windowing system, programmers have to be trained for the complex API under Windows.

- Windows OS and Mac systems have tough resource restrictions. For this reason, application programmers also have to be well trained in systems technology, so that they can optimize scarce resources.
- Increased network load: since the actual processing of the data takes place on the remote client, the data has to be transported over the network. As a rule, this leads to increased network stress.
- How to conduct transactions is controlled by the client. Advanced techniques like two-phase-committing can't be run.
- PCs are considered to be "untrusted" in terms of security, i.e. they are relatively easy to crack. Nevertheless, sensitive data is transferred to the PC, for lack of an alternative.
- Data is only "offered" on the server, not processed. Stored procedures are a form of assistance given by the database provider. But they have a limited application field and proprietary nature.
- Application logic can't be reused because it is bound to an individual PC- program.

- The influences on change-management are drastic: due to changes in business
- politics or law (e.g. changes in VAT computation) processes have to be changed. Thus possibly dozens of PC-programs have to be adapted because the same logic has been implemented numerous times. It is then obvious that in turn each of these programs has to undergo quality control because all programs are expected to generate the same results again.
- The 2-tier-model implies a complicated software-distribution procedure: as all of the application logic is executed on the PC, all those machines (maybe thousands) have to be updated in case of a new release. This can be very expensive, complicated, prone to error, and time-consuming. Distribution procedures include the distribution over networks (perhaps of large files) or the production of adequate media like CDs. Once it arrives at the user's desk, the software first has to be installed and tested for correct execution. Due to the distributed character of such an update procedure, system management cannot guarantee that all clients work on the correct copy of the program.

The 3-tier architecture endeavor to solve these problems. This goal is achieved primarily by moving the application logic from the client back to the server.

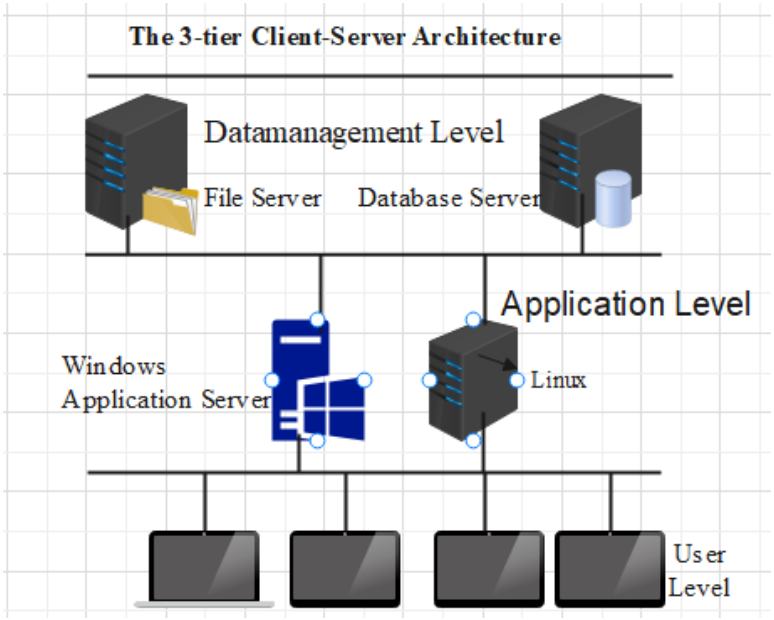
What are 3-tier and n-tier architecture?

The 3-tier architecture is illustrated in Figure 1.13.

- Client tier

This tier is responsible for the presentation of data, receiving user events, and controlling the user interface. The actual business logic (e.g. calculating added value tax) has been moved to an application server. Today, web server scripts such as PHP and JSP offer an alternative to traditionally written PC applications.

Figure 1.13 the 3-tier architecture



- Application Server-tier

This tier is new, i.e. it isn't present in 2-tier architecture in this explicit form. Business objects that implement the business rules "live" here, and are available to the client tier. This level now forms the central key to solving 2-tier problems. This tier protects the data from direct access by the clients.

The object-oriented analysis "OOA", on which many books have been written, aims in this tier: to record and abstract business processes in business objects. This way it is possible to map out the applications-server-tier directly from the CASE- tools that support OOA.

Furthermore, the term “component" is also to be found here. Today the term predominantly describes visual components on the client side. In the non-visual area of the

system, components on the server side can be defined as configurable objects, which can be put together to form new application processes.

- Data Server-tier

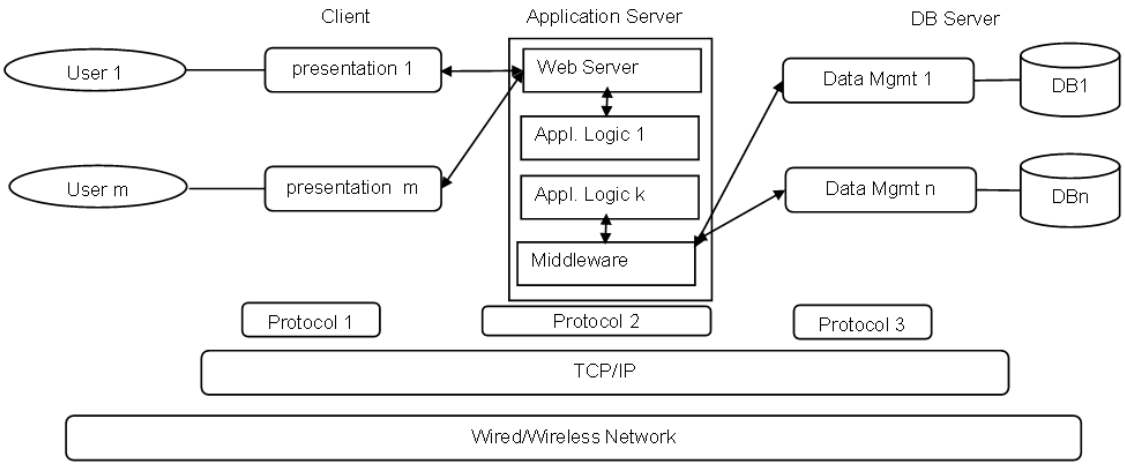
This tier is responsible for data storage. Besides the widespread relational database systems, existing legacy systems databases are often reused here.

It is important to note that boundaries between tiers are logical. It is quite easily possible to run all three tiers on one and the same (physical) machine. The main importance is that the system is neatly structured and that there is a well-planned definition of the software boundaries between the different tiers.

The advantage of 3-tier architecture

As previously mentioned, 3-tier architecture solves a number of problems that are inherent to 2-tier architectures. Naturally, it also causes new problems, but these are outweighed by the advantages. The software connection of the 3-tier model is shown in Figure 1.14, and the advantages are listed below.

Figure 1.14 the software connection of the 3-tier model



- Clear separation of user-interface control and data presentation from application logic. Through this separation, more clients are able to have access to a wide variety of server applications. The two main advantages for the client- applications are clear: quicker development through the reuse of pre-built business-logic components and a shorter test phase, because the server- components have already been tested.

- Re-definition of the storage strategy won't influence the clients. RDBMS offers certain independence from storage details for the clients. However, cases like changing table attributes make it necessary to adapt the client's application. In the future, even radical changes, like let's say switching from an RDBMS to an OODBS, won't influence the client. In well-designed systems, the client still accesses data over a stable and well-designed interface that encapsulates all the storage details.
- Business objects and data storage should be brought as close together as possible, ideally they should be together physically on the same server. This way - especially with complex accesses - network load is eliminated. The client only receives the results of a calculation - through the business object, of course.
- In contrast to the 2-tier model, where only data is accessible to the public, business objects can place applications-logic or "services" on the net. As an example, an inventory number has a "test digit", and the calculation of that digit can be made available on the server.
- As a rule servers are "trusted" systems. Their authorization is simpler than that of thousands of "untrusted" client PCs. Data protection and security is simpler to obtain. Therefore it makes sense to run critical business processes that work with security-sensitive data, on the server.
- Dynamic load balancing: if bottlenecks in terms of performance occur, the server process can be moved to other servers at runtime.
- Change management: of course, it's easy - and faster - to exchange a component on the server than to furnish numerous PCs with new program versions. For example, it is quite easy to run the new version of a tax object in such a way that the clients automatically work with the version from the exact date that it has to be run. It is, however, compulsory that interfaces remain stable and that old client versions are still compatible. In addition, such components require a high standard of quality control. This is because low quality components can, at worst, endanger the functions of a whole set of client applications. At best, they will still
  irritate the operator of the system.

Further, it is relatively simple to use wrapping techniques in 3-tier architecture. As implementation changes are transparent from the viewpoint of the object's client, a forward strategy can be developed to replace the legacy system smoothly. First, define the object's interface. However, the functionality is not newly implemented but reused from an existing

host application. That is, a request from a client is forwarded to a legacy system and processed and answered there.

In a later phase, the old application can be replaced by a modem solution. If it is possible to leave the business object's interfaces unchanged, the client application remains unaffected. A requirement for wrapping is, however, that a procedure interface in the old application remains existent. It isn't possible for a business object to emulate a terminal. It is also important for the project planner to be aware that the implementation of wrapping objects can be very complex.

DB Programming on the Web

Just as there is a diversity of programming languages available and suitable for conventional programming tasks, there is a diversity of languages available and suitable for Web DB programming. There is no reason to believe that any one language will completely monopolize the Web programming scene, although the varying availability and suitability of the current offerings are likely to favor some over others. Several Web programming languages are currently available to implement the application logic component in Figure 1.14.

Java is both available and generally suitable, but not all application developers are likely to prefer it over languages more similar to what they currently use, or, in the case of non-programmers, over higher level languages and tools. This is OK because there is no real reason why we must converge on a single programming language for the Web any more than we must converge on a single programming language in any other domain.

The Web does, however, place some specific constraints on our choices: the ability to deal with a variety of protocols and formats (e.g. graphics) and programming tasks; performance (both speed and size); safety; platform independence; protection of intellectual property; and the basic ability to deal with other Web tools and languages. These issues are not independent of one another. A choice that seemingly is optimal in one dimension may be sub-optimal or worse in another.

- **CGI (Common Gateway Interface)**

A Web daemon executes a CGI program on the server and returns the results to the client (e.g. a query against a database server), rather than simply returning a copy of the referenced file, as it does with an HTML reference. Parameters are passed from the server to the CGI program as environment variables.

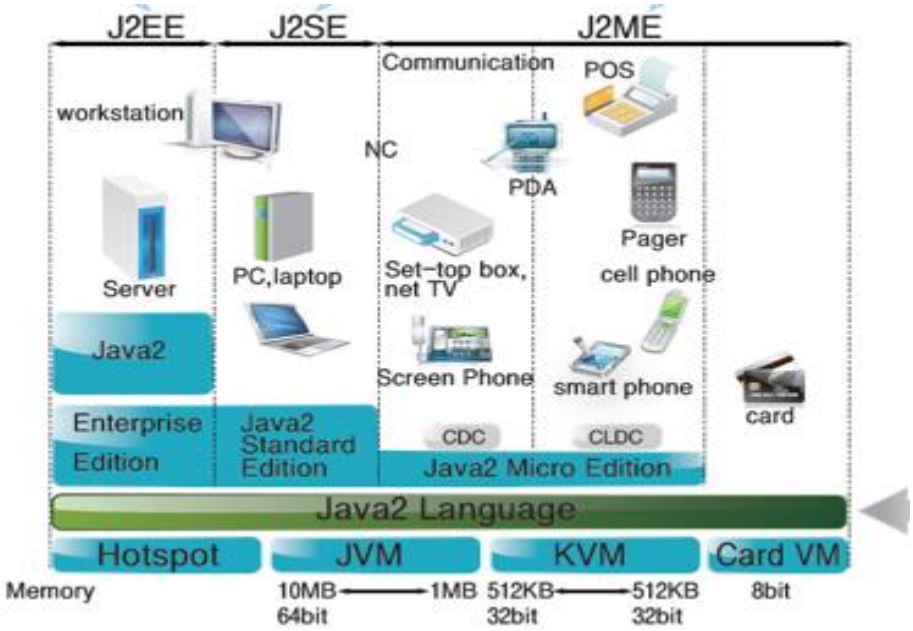
The program is sometimes written in C, C++, or some other compiled programming language, but more often it is written in a scripting language (e.g. Perl, Tel, sh). To prevent damage to the server, CGI programs generally are stored in a protected directory under the exclusive control of the webmaster.

- **Java**

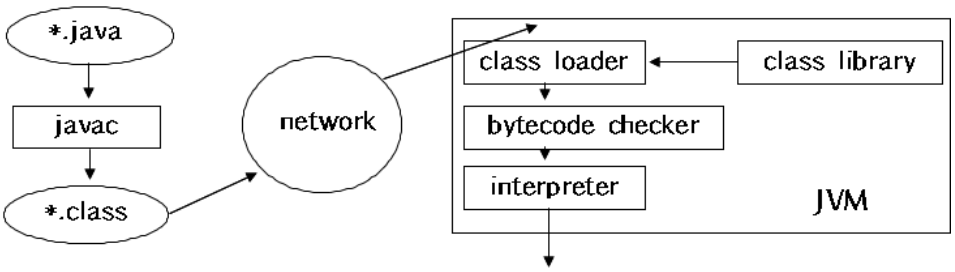
Java is the leading contender for a full feature programming language targeted at Internet applications. It's advantages are: familiarity (derived from C++), platform independence (will run on any platform which implements the Java Virtual Machine), performance (byte-code compiled faster than fully interpreted), and safety (downloaded applets are checked for integrity, and only interpreted by trusted Virtual Machine).

Java is being aggressively distributed and promoted by Sun Microsystems (Oracle Co.), which developed it, and, evidently, sees it as a way to loosen Microsoft's and Intel's grip on the computer platform. The leading web browsers, now includes the Java VM, and JavaScript/JSP are appearing on web sites everywhere. Even Microsoft, which is promoting Visual Basic Script for this purpose, has licensed Java from Sun and will be supporting it in its browsers.

The list of Java licensees is long, and includes other major players, such as IBM. Sun (Oracle Co.), is distributing a Java developer's kit free of charge as of this writing, in the interest of promoting Java's widespread use. It recently announced the development of microprocessors optimized for Java for different markets (from cellular phones to high performance 3D "Network Appliances". If their strategy is successful, the application platform is raised, and Java displaces Windows or other OS's as the target platform of application developers, then the whole ballgame changes, and the impact is potentially across the entire computer industry, not just the Internet.



The ability to deliver a platform-independent application, or, more correctly, an OS-independent application, is of great appeal to developers, who spend a large portion of their resources developing and maintaining versions off their products for the different hardware/software platform combinations. With Java, one set of sources, and, even more important, one byte-compiled executable, can be delivered for all hardware/software platforms.



While the interpretation of a byte-compiled program is slower than the execution of a native executable, the claim is made that, once interpreted, the resulting executable is of comparable performance, which means Java applications could be interpreted once and the result cached locally and thereafter executed optimally. This is great news for Unix, OS/2, and Macintosh vendors and users, who often suffer from limited or delayed availability of software and high prices due to limited demand, and, likewise, for non-Intel chip and computer vendors.

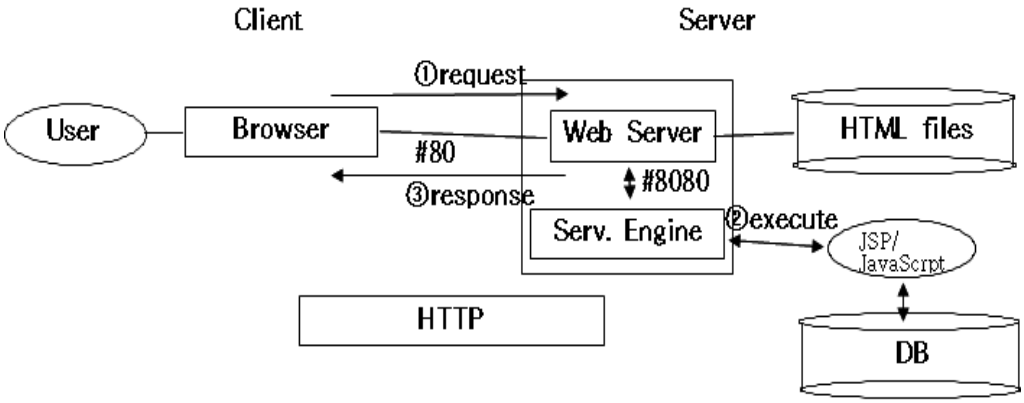
It is potentially disastrous news for Microsoft and Intel, who, arguably, often sell their products solely based on their market position, rather than their technical merit. Hopefully, the result will be a more level playing field for vendors and more choices for consumers and not just the replacement of Microsoft and Intel with Sun (Oracle Co.).

That said, not everyone agrees that Java is the answer. The most common complaint is that Java is not simple; it is a slimmed-down, cleaned-up C++, with a big GUI library. C++ programming is not described by most as "simple", and Java programming is not much simpler, especially when compared to HTML, or some other languages put forward as its competition. Java is the market leader at the moment, so it is the obvious target.

- **JavaScript**

JavaScript is Netscape's scripting language for integrating HTML, Netscape plug-ins, and Java applets. It is based on Java, and is mostly syntactically compatible, but differs from Java in that it is interpreted, rather than compiled, only supports built certain built-in objects and user-defined functions, rather than full support for user-defined classes with inheritance and methods, and is integrated with HTML, rather than invoked from HTML files, weakly typed, and dynamically bound.

JavaScript is meant to extend HTML to be more of a full programming language while retaining HTML's ease of use. The principal criticism of Java programming is that much more complex than HTML programming, more like C++ programming, and therefore is not as accessible to users as HTML. This is an issue that JavaScript attempts to address.



- **Python**

Python is an interpreted, object-oriented language developed as a full-featured, but easy-to-use, scripting language, by Guido van Rossum at CWI in the Netherlands. Initially developed in a Unix environment, Python is now available on PCs and Macs, and applications are portable across platforms.

Python has developed a substantial, although still modest, following, as a scripting language, an application development language, and an embedded extension language. Python's design was most influenced by ABC, a little-known language also developed at CWI. Python's syntax evokes C and C++ but doesn't stick too closely to those languages.

Python fans tout its clear, intuitive syntax in comparison to C, C++, Java, Perl, shell languages, and most other interpreted languages, the completeness of its type system and its suitability for significant application development in comparison to Tcl, and its extensibility with Python and C/C++ libraries. Like Java, Perl and Tcl, Python offers a portable GUI library, several.

Perl advocates complain about the lack of regular expression matching and output formatting natively in Python. Perl is a little more of a sysadmin's shell language than Python, and Tcl is a little simpler and less capable. Python is more of a regular programming language, but simpler and easier to program than Java. But, all are suited to Internet programming.

- **VBScript (Visual Basic Script)**

VBScript is Microsoft's planned candidate for an Internet scripting language. It is a subset of Visual Basic, Microsoft's popular visual programming language, with no GUI building capability, unsafe operations removed, and with access to other applications via OLE.

VBScript source code is embedded in HTML and downloaded to the client in the HTML file, where it is compiled and executed in association with its runtime libraries. Microsoft envisions an OLE Scripting Manager on the client side with which browsers interact with a specified interface. The Scripting Manager would manage the compilation and invocation of downloaded scripts in Visual Basic Script or any other scripting language.

Microsoft also intends to support Visual Basic and Java in this way. The idea is to make multiple language runtimes pluggable into browsers. Microsoft intends to elicit the cooperation of various consortia and vendors in defining and standardizing this interface. Microsoft intends to support VBScript on its various Windows platforms and the Macintosh and will license the technology to UNIX vendors.

- **PHP (PHP Hypertext Pre-processor) Server Side Script**

In the early days of the spread of the Internet, websites were created using Perl language and C language. However, the problem with the Perl language and the C language is that it is inefficient and very difficult to develop. The biggest problem with the Perl language and the C language is the processing method. Each time a client connects one by one, a process is created. Therefore, as the number of connections increases, overload occurs more frequently, and the server often goes down.

Therefore, various Web script languages have been developed to supplement the above problems and provide faster service. Microsoft's ASP, Sun's JSP, and Zend's PHP are famous Web scripting languages. In the above three languages, when a client connects one by one, it creates a process first and then creates a thread in it to respond. Since tasks are processed in units of threads, the load on the server is reduced. PHP is developed in the C language, so its syntax is similar to C and its execution speed is also fast.

Because PHP is free, it does not cost money to use, and it is independent of the platform by supporting multiple OSs such as Windows, Linux, and Unix. Also, it runs a bit faster on Linux, a free OS, and is called APM (Apache/PHP/MySQL) because it works well with MySQL, a free RDBMS. Usually, PHP works in conjunction with a web server called Apache. When a user requests a web document, Apache processes the HTML document, and PHP processes the PHP Script.

Summary

The efficient storage and manipulation of data is the major goal of the database management system. A database is an organized collection of related data. Only data that has been processed provides meaningful information that enables an organization to make critical decisions. Large data files were stored, processed, and retrieved using computer file processing systems before DBMS. Computer file processing systems have limitations such as data duplications, limited data sharing, and no program data independence. A database technique was created to get around these restrictions. Program data independence, improved data sharing, and reduced data redundancy are the major benefits of the DBMS method. This chapter has provided a general overview of DBMS as well as its historical development. The responsibilities of a Database administrator, two-tier, and three-tier architecture were analyzed in this chapter.

Questions

1. What is the data?
2. What is the information?
3. What are the different between data and information?
4. What are the disadvantages of the file processing system?
5. What is the Database? Give an example
6. What is the DBMS?
7. What are the advantages of the DBMS?

Exercises

1. Please look at figure 1.13 the 3-tier architecture from the textbook and a summary of its processing.
2. Please summary of the role of a database administrator (DBA).
3. Please draw about 2-tier client/server architecture and 3-tier client/ server architecture and explain in brief.
4. Please explore on the internet to find the various categories of the data model and explain them in brief.

Chapter 02

Relational Model

This chapter is dedicated to the relational model which is in use since the late 1970s by E.F. Codd (Edgar Frank Codd). Various operations in relational algebra and relational calculus are given in this chapter. After completing this chapter, the reader should be familiar with the following concepts of evolution and the importance of the relational model, terms in the relational model like tuple, domain, cardinality, and degree of relation, and operations in relational algebra and relational calculus, the difference between relational algebra and relational calculus.

In this chapter, you will learn:

2.1. Overview

The relational model for large shared data is written by E.F. Codd (Edgar Frank Codd) of IBM in 1970.

The first database systems were based on network and hierarchical models. The relational model was first proposed by E.F. Codd in 1970 and the first such systems (notably INGRES and System/R) were developed in the 1970s. The relational model is now the dominant model for commercial data processing applications.

A relational database consists of a collection of tables, each having a unique name. A row in a table represents a relationship among a set of values. Thus, a table represents a collectin of relationships. There is a direct correspondence between the concept of a table and the mathematical concept of a relation. A substantial theory has been developed for relational databases.

The text uses long attribute names instead of abbreviation words in the notes as follows.

- customer-name instead of cname
- customer-city instead of ccity
- branch-city instead of bcity
- branch-name instead of bname
- account-number instead of account#
- loan-number instead of loan#
- banker-name instead of banker

The terms commonly used by the user, model, and programmers are given below:

User	**Model**	**Programmer**
Row	Tuple	Record
Column	Attribute	Field
Table	Relation	File

2.2. Relation

Mathematicians define a relation to be a subset of a Cartesian product of a list of domains. Mathematic terms 'relation' and 'tuple' is placed of 'table' and 'row' that are used for the relational model.

We can add, delete and modify rows to reflect changes in the real world. A row of a table will consist of an n-tuple where n is the number of attributes. Actually, the table contains

a subset of the set of all possible rows. We refer to the set of all possible rows as the Cartesian product of the sets of all attribute values.

- Figure 2.1 shows the Deposit and Customer tables for our banking example
- The deposit table has four attributes.
- For each attribute there is a permitted set of values, called the domain of that attribute.
- E.g. the domain of bname is the set of all branch names.

Figure 2.1 the Deposit and Customer relations

bname	account#	cname	balance
Downtown	101	Johnson	500
Lougheed Mall	215	Smith	700
SFU	102	Hayes	400
SFU	304	Adams	1300

cname	street	ccity
Johnson	Pender	Vancouver
Smith	North	Burnaby
Hayes	Curtis	Burnaby
Adams	No.3 Road	Richmond
Jones	Oak	Vancouver

Let D1 denote the domain of bname, and D2, D3, and D4 the remaining attributes' domains respectively. We may denote this as

$$D, X D2 X D3 X D4$$

for the deposit table, where D1, D2, D3, and D4 denote the set of all branch names, all account numbers, all customer names and all balances, respectively.

In general, for a table of n columns, we may denote the Cartesian product of D1, D2 .... , Dn by

$$x^{n}_{i=1} D_i$$

Then, any row of the deposits consisting of a four-tuple (v1, v2, V3, V4)

$$v1 \in D1, v2 \in D2, v3 \in D3, v4 \in D4$$

In general, a deposit contains a subset of the set of all possible rows. That is, a deposit is a subset of

$$D1 \times D2 \times D3 \times D4 \text{ or abbreviated to } x^{4}_{i=1} Di$$

In general, a table of n columns must be a subset of

$$x^{n}_{i=1} D_i \text{ (all possible rows)}$$

Mathematicians define a relation to be a subset of a Caresian product of a list of domains. You can see the correspondence with our tables. The mathematical terms 'relation' and 'tuple' in place of 'table' and 'row' are used for the relational model.

Some more formalities:

- Let the tuple variable t refer to a tuple of the relation R.
- We say t∈R to denote that the tuple t is in relation to R.

For the first tuple t in relation to deposit,

- Then t[bname] = t[1] = the value of t on the bname attribute.
- So t[bname] = t[1] = "Downtown",
- and t[cname] = t[3] = "Johnson".

We'll also require that the domains of all attributes be indivisible units.

- A domain is atomic if its elements are indivisible units.
- For example, the set of integers is an atomic domain.
- The set of all sets of integers is not.
- Why? Integers do not have subparts, but sets do, the integers comprising them
- We could consider integers non-atomic if we thought of them as ordered lists of digits.

2.3. Database Scheme

The difference between logical design (a database scheme) and structure of a database file (database instance). The relation between all attributes and domains is called a relation scheme.

We distinguish between a database scheme (logical design) and a database instance (data in the database at a point in time). A relation scheme is a list of attributes and their corresponding domains. For example, the relation scheme for the deposit relation:

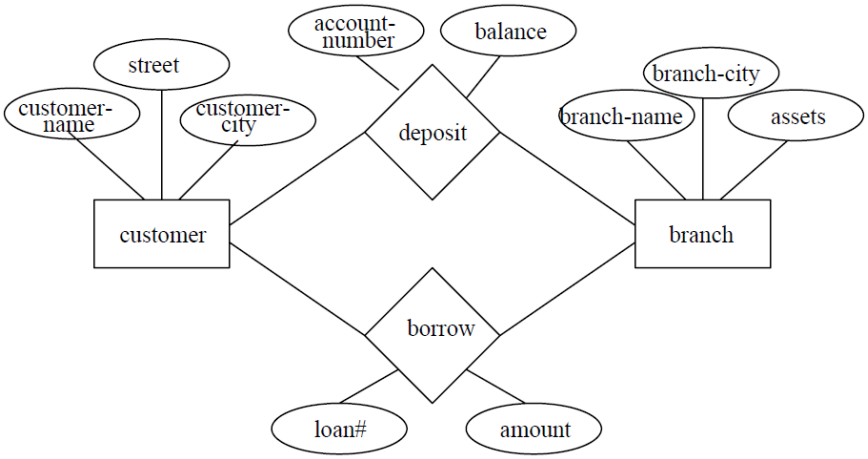
Deposit-scheme= (bname, account# , cname, balance)

We may state that deposit is a relation on scheme Deposit-scheme by writing deposit(Deposit-scheme). If we wish to specify domains, we can write:

(bname: string, account#: integer, cname: string, balance: integer).

Note that customers are identified by name. In the real world, this would not be allowed, as two or more customers might share the same name. Figure 2.2 shows the E-R diagram for a banking enterprise.

Figure 2.2 E-R diagram for the banking enterprise



The relation schemes for the banking example used throughout the text are:

Branch-scheme = (bname, assets, bcity)
Customer-scheme= (cname, street, ccity)
Deposit-scheme= (bname, account#, cname, balance) Borrow-scheme = (bname, loan#, cname, amount)

Some attributes appear in several relation schemes (e.g. bname, cname). This is legal and provides a way of relating tuples of distinct relations. Why not put all attributes in one relation?

Suppose we use one large relation instead of customer and deposit:

Account-scheme= (bname, account#, cname, balance, street, ccity)

- If a customer has several accounts, we must duplicate her or his address for each account.
- If a customer has an account but no current address, we cannot build a tuple, as we have no values for the address.
- We would have to use null values for these fields.
- Null values cause difficulties in the database.
- By using two separate relations, we can do this without using null values

2.4. Keys

A super key is a set of attributes that uniquely identify each tuple of a relation. A candidate key is a super key with no redundant attributes known as a candidate key i.e should not contain any column that contains duplicate data. A primary key is a specific choice of a minimal set of attributes (columns) that uniquely specify a tuple (row).

More formally, if we say that a subset k of a relation R is a superkey for R, we are restricting consideration to relations r(R) in which no two distinct tuples, t1 and t2, have the same values on all attributes ink. In other words,

- If t1 and t2 are in r, and $t1 \neq t2$, then $t1[\ k\ ] \neq t2[\ k\ ]$.

The notions of super key, candidate key, and primary key all apply to the relational model. For example, in Branch-scheme,

- {bname} is a superkey.
- {bname, bcity} is a superkey.
- {bname, bcity} is not a candidate key, as the superkey {bname} is contained in it.
- {bname} is a candidate key.
- {bcity} is not a superkey, as branches may be in the same city.
- We will use {bname} as our primary key. The primary key to Customer-scheme is {cname}.

2.5. Relational Algebra

Requesting information from a database by the user is used query language. Relational Algebra was procedural is used in this language. There are a lot of operations in this language such as select, project, rename, Cartesian product, union, and set-difference. Moreover, there are additional functions as well like set-intersection, natural join, division, and assignment.

A query language is a language in which a user requests information from a database. These are typically higher-level than programming languages. They may be one of:

- Procedural, where the user instructs the system to perform a sequence of operations on the database. This will compute the desired information.
- Nonprocedural, where the user specifies the information desired without giving a procedure for obtaining the information.

A complete query language also contains facilities to insert and delete tuples as well as to modify parts of existing tuples. Relational algebra is a procedural query language, and operations produce a new relation as a result.

Six fundamental operations:

- select (unary): σ_p (R) (P a predicate)
- project (unary): π_s (R) (S a list of attributes)
- rename (unary): ρ_x (R) (X a relation name)
- Cartesian product (binary): RxS
- union (binary): RυS
- set-difference (binary): R-S

Several other operations are defined in terms of the fundamental operations:

- set-intersection: R$\cap$S
- natural join: R∞S
- division: R$\div$S
- assignment: R$\leftarrow$S

2.5.1. Fundamental Operations

Select

The select operation selects tuples that satisfy a given predicate. Select is denoted by a lowercase Greek sigma (ϭ), with the predicate appearing as a subscript. The argument relation is given in parentheses following the ϭ.

Let Figure 2.3 be the Borrow and Brach relations in the banking operations

Figure 2.3 the Borrow and Brach relation

bname	account#	cname	balance
Downtown	101	Johnson	1000
Lougheed Mall	215	Smith	2000
SFU	102	Hayes	1500

bname	assets	bcity
Downtown	9,000,000	Vancouver
Lougheed Mall	21,000,000	Burnaby
SFU	17,000,000	Burnaby

For example, to select tuples (rows) of the borrow relation where the branch is "SFU", we would write

$$\sigma_{bname=' SFU'}(Borrow)$$

The new relation created as the result of this operation consists of one tuple:

(SFU, 15, Hayes, 1500) .

We allow comparisons using =, ≠, <, ≤, >, and ≥ in the selection predicate. We also allow the logical connectives v (or) and/\ (and). For example:

$$\sigma_{bname=' Downtown ' \wedge \ amount>1200}(Borrow)$$

Suppose there is one more relation, Client, shown in Figure 3.4, with the scheme Client_scheme = (cname, banker)

We might write to find clients who have the same name as their banker.

$$\sigma_{cname=banker}(Client)$$

Figure 2.4 the client relation

cname	Backer
Hayes	Jones
Johnson	Johnson

Project

The project operation copies its argument relation for the specified attributes only. Since a relation is a set, duplicate rows are eliminated. Projection is denoted by the Greek capital letter pi (IT). The attributes to be copied appear as subscripts.

For example, to obtain a relation showing Customers and Branches, but ignoring amount and loan#, we write

$$\pi_{bname,\ cname}\ (Borrow)$$

We can perform these operations on the relations resulting from other operations. To get the names of customers having the same name as their bankers,

$$\pi_{cname}\ (\sigma_{\ cname=banker}\ (Client))$$

Think of selecting as taking rows of a relation, and project as taking columns of a relation.

Cartesian Product

The Cartesian product operation of two relations is denoted by a cross (x), written

R_1 x R_2 for relations R_1 and R_2

The result of R_1 x R_2 is a new relationship with a tuple for each possible pairing of tuples from R_1 and R_2. In order to avoid ambiguity, the attribute names have attached to them the name of the relation from which they came. If no ambiguity will result, we drop the relation name. The result Client x Customer is a very large relation as shown in Table 2.1.

Table 2.1 *the Client x Customer relation*

Client.cname	banker	Customer.cname	street	city
Hayes Johnson	Jones Johnson	....	....	....

If R_1 has n_1 tuples, and R_2 has n_2 tuples, then $R = R_1$ x R_2 will have $n_1 \cdot n_2$ tuples. The resulting scheme is the concatenation of the schemes of R_1 and R_2, with relation names added as mentioned.

To find the clients of banker Johnson and the city in which they live, we need information on both Client and Customer relations. We can get this by writing

$$\sigma_{\ banker\ =\ '\ Johnson\ '}\ (Client\ X\ Customer)$$

However, the Customer.cname column contains customers of bankers other than Johnson. Therefore, we want rows where Client.cname = Customer.cname. So we can write

$$\sigma_{\text{banker = ' Johnson' } \wedge \text{ Client.cname = Customer.cname}} (\text{Client X Customer})$$

to get just these tuples.
Finally, to get just the customer's name and city, we need a projection:

$$\pi_{\text{client.cname, ccity}}(\sigma_{\text{banker = ' Johnson' } \wedge \text{ Client.cname = Customer.cname}} (\text{Client X Customer}))$$

Rename

The rename operation solves the problems that occur with naming when performing the Cartesian product of relation with itself. Suppose we want to find the names of all the customers who live on the same street and in the same city as Smith. We can get the street and city of Smith by writing

$$\pi_{\text{street,ccity}}(\sigma_{\text{cname = 'Smith'}} (\text{Customer}))$$

To find other customers with the same information, we need to reference the customer relation again:

$$\sigma_{p} (\text{Customer X } (\pi_{\text{street, ccity}} ((\sigma_{\text{cname = ' Smith'}} (\text{Customer}))))$$

where P is a selection predicate requiring street and ccity values to be equal.

Problem: how do we distinguish between the two street values appearing in the Cartesian product, as both come from the Customer relation?

Solution: use the rename operator, denoted by the Greek letter rho (p).

We write

$$Px(R)$$

to get the relation R under the name of X.

If we use this to rename one of the two Customer relations we are using , the ambiguities will disappear.

$$\pi_{\text{Customer.cname}}(\sigma_{\text{Cust2.street}= \text{ Customer.street } \wedge \text{ Cust2.ccity } = \text{ Customer.ccity}})$$
$$(\text{Customer x } (\pi_{\text{street.ccity}}(\sigma_{\text{cname='Smith'}} (P_{\text{Cust2}} (\text{Customer}))))$$

Union

The union operation is denoted u as in set theory. It returns the union (set union) of two compatible relations. For a union operation RUS to be legal , we require that:

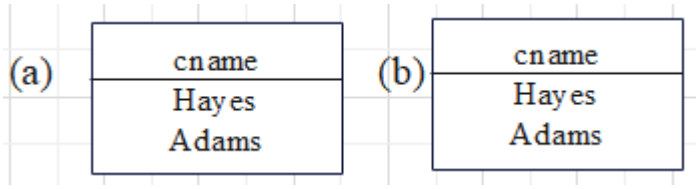
- R and S must have the same number of attributes.
- The domains of the corresponding attributes must be the same.

To find all customers of the SFU branch, we must find everyone who has a loan or an account or both at the branch. We need both Borrow and Deposit relations for this:

$$\pi_{\text{cname}} (\sigma_{\text{bname = ' SFU'}} (\text{Borrow})) \cup \pi_{\text{cname}} (\sigma_{\text{bname = ' SFU'}} (\text{Deposit}))$$

As in all set operations, duplicates are eliminated, giving the relation of Figure 2.5(a).

Figure 2.5 the union and set difference operations



Set Difference

The set difference operation is denoted by the minus sign (-). It finds tuples that are in one relation, but not in another. Thus R-S results in a relation containing tuples that are in R but not in S.

To find customers of the SFU branch who have an account there but no loan, we write

$$\pi_{cname}(\sigma_{bname = 'SFU'}(Borrow)) - \pi_{cname}(\sigma_{bname = 'SFU'}(Deposit))$$

The result is shown in Figure 2.5(b). We can do more with this operation. Suppose we want to find the largest account balance in the bank.

Strategy:

- Find a relation R containing the balances not the largest.
- Compute the set difference of R and the Deposit relation.

To find, we write

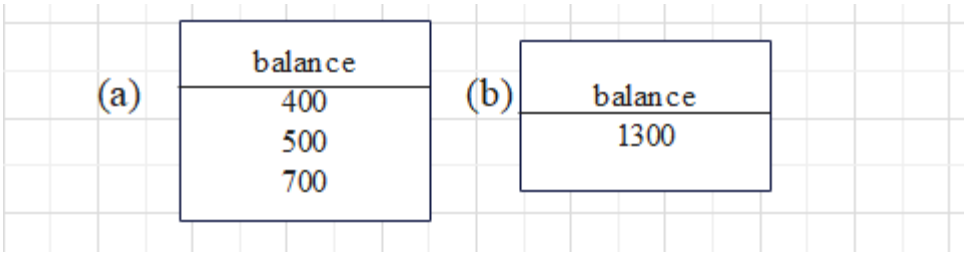
$$\pi_{Deposit.balance}(\sigma_{Deposit.balance < D.balance}(Deposit \times \rho_{D}(Deposit))$$

This resulting relation contains all balances except the largest one. (See Figure 2.6(a)). Now we can finish our query by taking the set difference:

$$\pi_{Deposit.balance}(\sigma_{Deposit.balance < D.balance}(Deposit \times \rho_{D}(Deposit))$$

Figure 2.6(b) shows the result.

Figure 2.6 find the largest account balance in the bank



2.5.2. Additional Operations

Additional operations are defined in terms of the fundamental operations. They do not add power to the algebra, but are useful to simplify common queries.

Intersection

The set intersection operation is denoted by n and returns a relation that contains tuples that are in both of its argument relations. It does not add any power as

$$R \cap S = R - (R - S)$$

To find all customers having both a loan and an account at the SFU branch, we write

$$\pi_{cname}(\sigma_{bname = 'S FU'}(Borrow)) \cap \pi_{cname}(\sigma_{bname = 'S FU'}(Deposit))$$

Natural Join

Often we want to simplify queries on a Cartesian product. For example, to find all customers having a loan at the bank and the cities in which they live, we need Borrow and Customer relations:

$$\pi_{Borrow\ cname,ccity}(\sigma_{Borrow.cname = Customer.cname}(Borrow\ X\ Customer))$$

Our selection predicate obtains only those tuples pertaining to only one cname. This type of operation is very common, so we have the natural join operation, denoted by a ∞ sign. Natural join combines a Cartesian product and a selection into one operation. It performs selection-forcing equality on those attributes that appear in both relation schemes. Duplicates are removed as in all relation operations.

To illustrate, we can rewrite the previous query as

$$II_{cname,\ ccity}(Borrow \infty Customer)$$

The resulting relation is shown in Figure 2.7.

Figure 2.7 Joining Borrow and Customer relations

cname	ccity
Smith	Burnaby
Hayes	Burnaby
Jones	Vancouver

We can now make a more formal definition of natural join.

- Consider r and s to be sets of attributes.
- We denote attributes appearing in both relations by r ∩ s.
- We denote attributes in either or both relations by r U s.

Consider two relations R(r) and S(s).

- The natural join of Rand S, denoted by R∞S is a relation on the scheme rUs.
- It is a projection onto R U S of a selection on R x S where the predicate requires $R.A_i=S.A_i$ for each attribute A_i in r∩s.

Formally,

$$R \infty S = \pi_{rUs} (\sigma_{R.A1} = S.A1 \wedge \ldots \wedge R.An = S.An (R \times S))$$

Where r∩s = {A1,…….An}.

To find the assets and names of all branches which have depositors living in Stamford, we need Customer, Deposit and Branch relations:

$$\pi_{bname,\ assets} (\sigma_{ccity='Standford'} (Customer \infty Deposit \infty Branch))$$

Note that ∞ is associative. To find all customers who have both an account and a loan at the SFU branch:

$$\pi\ cname\ (\ \sigma\ bname='SFU'\ (Borrow \infty Deposit))$$

This is equivalent to the set intersection version we wrote earlier. We see now that there can be several ways to write a query in relational algebra. If two relations R(r) and S(s) have no attributes in common, then r∩s =, and R∞ S = R X S.

Division

The division operation, denoted ÷, is suited to queries that include the phrase "for all". Suppose we want to find all the customers who have an account at all branches located in Brooklyn.

Strategy: think of it as three steps.

1. We can obtain the names of all branches located in Brooklyn by

$$R_1 = \pi_{bname} (\sigma_{bcity='Brooklyn'} (Branch))$$

2. We can also find all cname, bname pairs for which the customer has an account by

$$R_2 = \pi_{\text{cname, bname}} (\text{Deposit})$$

3. Now we need to find all customers who appear in R_2 with every branch name in R_1

The divide operation provides exactly those customers:

$$\pi_{\text{cname, bname}} (\text{Deposit}) \div \pi_{\text{bname}}(\sigma_{\text{bcity} = \text{' Brooklyn '}} (\text{Branch}))$$

which is simply $R_2 \div R_1$

Formally,

- Let R(r) and S(s) be relations.
- Let s U r.
- The relation R + S is a relation on scheme r - s.
- A tuple t is in R ÷ S if for every tuple t_S in S there is a tuple t_R in R satisfying both of the following

$$t_R[s] = t_S[s]$$
$$t_R[r - s] = t[r - s]$$

- These conditions say that the r - s portion of a tuple t is in R ÷ S if and

only if there are tuples with the r - s portion and the s portion in R for every value of the s portion in relation S.

We will look at this explanation in class more closely. The division operation can be defined in terms of the fundamental operations.

$$R \div S = \pi_{r-s} (R) – \pi_{r-s} ((\pi_{r-s} ((R) \times S) - R)$$

Assignment

Sometimes it is useful to be able to write a relational algebra expression in parts using a temporary relation variable (as we did with R_1 and R_2 in the division example). The assignment operation, denoted +-, works like an assignment in a programming language.

We could rewrite our division definition as

$$T_1 \longleftarrow \pi_{r-s} (R)$$
$$T_1 \longleftarrow \pi_{r-s} ((T_1 \times S) - R)$$

No extra relation is added to the database, but the relation variable created can be used in subsequent expressions. Assignment to a permanent relation would constitute a modification to the database.

2.5.2. Modifying the Database

Up until now, we have looked at extracting information from the database. We also need to add, remove and change information. Modifications are expressed using the assignment operator.

Deletion

Deletion is expressed in much the same way as a query. Instead of displaying, the selected tuples are removed from the database. We can only delete whole tuples.
In relational algebra, a deletion is of the form

$$r \longleftarrow r\text{-}E$$

where r is a relation and E is a relational algebra query. Tuples in r for which E is true are deleted.
Some examples:

1. Delete all of Smith's account records.

$$Deposit \longleftarrow Deposit - \sigma_{cname='Smith'}(Deposit)$$

2. Delete all loans with loan numbers between 1300 and 1500.

$$Deposit \longleftarrow Deposit - \sigma_{loan\# >= 1300" loan\# =< 1500}(Deposit)$$

3. Delete all accounts at branches located in Needham.

$$r_1 \longleftarrow \sigma_{bcity='Needham'}(Deposit \infty Branch)$$
$$r_2 \longleftarrow \prod_{bname,\ account\#,cname,\ balance}(r_1)$$
$$Deposit \longleftarrow Deposit - r_2$$

Insertion

To insert data into a relation, we either specify a tuple or write a query whose result is the set of tuples to be inserted. Attribute values for inserted tuples must be members of the attribute's domain.

An insertion is expressed by $r \longleftarrow r \cup E$

where r is a relation and E is a relational algebra expression.
Some examples:

1. To insert a tuple for Smith who has $1200 in account 9372 at the SFU branch.

   Deposit Deposit $\cup$ {('SFU' , 9372, 'Smith', 1200)}

2. To provide all loan customers in the SFU branch with a $200 savings

Account.

$r_1 \leftarrow \sigma_{bname='SFU'}$ (Borrow)

$r_2 \leftarrow \pi_{bname,loan\#,cname}(r1)$

Deposit $\leftarrow$ Deposit $\cup$ (r_2 x {(200)})

Update

Updating allows us to change some values m a tuple without necessarily changing all.

We use the update operator, γ, with the form

$$\gamma_{A \leftarrow E}(R)$$

where r is a relation with attribute A, which is assigned the value of expression E. The expression E is any arithmetic expression involving constants and attributes in a relation.

Some examples:

1. To increase all balances by 5 percent:

$$\gamma_{balance \leftarrow balance * 1.05}(R)$$

This statement is applied to every tuple in Deposit.

2. To make two different rates of interest payment, depending on the balance account.

$$\gamma_{balance \leftarrow balance*1.06}(\sigma_{balance > 10000}(Deposit))$$
$$\gamma_{balance \leftarrow balance*1.05}(\sigma_{balance \leq 10000}(Deposit))$$

Note: in this example the order of the two operations is important.

2.6. The Tuple Relational Calculus

The relational calculus is nonoperational. The users define queries in terms of what they want but not in terms of what the computer cannot compute.

The tuple relational calculus is a nonprocedural language. (The relational algebra was procedural.) We must provide a formal description of the information desired.

A query in the tuple relational calculus is expressed as

$$\{t \mid P(t)\}$$

i.e . the set of tuples t for which predicate P is true.

We also use the notation

- t[a] to indicate the value of tuple ton attribute a.
- t ∈ r to show that tuple t is in relation r.
-

Several tuple variables may appear in a formula. A tuple variable is said to be a free variable unless it is quantified by a ∃ or a ∀. Then it is said to be a bound

A formula is built of atoms. An atom is one of the following forms:

- s ∈ r, where s is a tuple variable, and r is a relation (is not allowed).

- s[x]θ u[y], where and u are tuple variables, and s and y are attributes, and θ is a comparison operator (<, ≤, =, ≠, >, ≥).
- s[x] θ c, where c is a constant in the domain of attribute x.

Formulae are built up from atoms using the following rules:

- An atom is a formula.
- If P is a formula, then so are ¬P and (P).
- If P_1 and P_2 are formulae, then so are P_1 v P_2, P_1 ∧ P_2 and $P_1 \Rightarrow P_2$
- If P(s) is a formula containing a free tuple variable s, then the followings are formulae also.

$$\exists s \in r(P(s)) \text{ and } \forall s \in r(P(s))$$

Note some equivalences:

- $P_1 \wedge P_2 = \neg (\neg P_1 \vee \neg P_2)$
- $\forall t \in r(P(t)) = \neg \exists t \in r(\neg P(t))$
- $P_1 \Rightarrow P_2 = \neg P_1 \vee P_2$

For example, to find the branch name, loan number, customer name and amount for loans over $1200:

$$\{t \mid t \in Borrow \wedge t[amount] > 1200\}$$

This gives us all attributes, but suppose we only want the customer names. (We would use project in the algebra.) We need to write an expression for a relation on scheme (cname).

$$\{t \mid \exists s \in Borrow (t[cname]=s[cname] \wedge s[amount] > 1200)$$

In English, we may read this equation as "the set of all tuples t such that there exists a tuple s in the relation Borrow for which the values of t and s for the cname attribute are equal, and the value of s for the amount attribute is greater than 1200." The notation ∃t ∈ r(Q[t]) means "there exists a tuple t in relation r such that predicate Q[t] is true".

How did we get the above expression? We needed tuples on scheme cname such that there were tuples in Borrow pertaining to that customer name with amount attribute. The tuples t get the scheme cname implicitly as that is the only attribute t is mentioned with.

Let's look at a more complex example. "Find all customers having a loan from the 'SFU' branch, and the the cities in which they live."

$$\{t \mid \exists s \in Borrow (t[cname]=s[cname] \wedge s[bname] = 'SFU' \wedge \exists u \in Customer(u[cname]=s[cname] \wedge t[ccity] = u[ccity]))\}$$

In English, we might read this as "the set of all (cname, ccity) tuples for which cname is a borrower at the 'SFU' branch, and ccity is the city of cname". Tuple variable s ensure that the customer is a borrower at the 'SFU'. branch. Tuple variable u is restricted to pertain to the same customer as s, and also ensures that ccity is the city of the customer.

The logical connectives ∧ (AND) and v(OR) are allowed, as well as ¬ (negation). We also use the existential quantifier ∃ and the universal quantifier ∀.

Some more examples:

1. Find all customers having a loan, an account, or both at the SFU branch:

{t | ∃s ∈ Borrow (t[cname]=s[cname] ∧ s[bname] = 'SFU')
v ¬∃u ∈ Deposit(t[cname]=u[cname] ∧ u[bname] ='SFU')}

Note the use of the v connective. As usual, set operations to remove all duplicates.

2. Find all customers who have both a loan and an account at the SFU branch. Solution: simply change the v connective in 1 to a ∧.
3. Find customers who have an account, but not a loan at the SFU branch.

{t | ∃u ∈ Deposit(t[cname]=u[cname] ∧ u[bname] ='SFU')
v ¬∃s ∈ Borrow(t[cname]=s[cname] ∧ s[bname] ='SFU')}

4. Find all customers who have an account at all branches located in Brooklyn.

(We used division in relational algebra.) For this example, we will use implication, denoted by a pointing finger in the text, but by ⇒ here. The formula P ⇒ Q means P implies Q, or, if P is true, then Q must be true.

{t I ∀u ∈ Branch(u[bcity] = 'Brooklyn' ⇒
¬∃s ∈ Borrow(t[cname]=s[cname] ∧ u[bname] =s[bname]))}

In English: "the set of all cname tuples t such that for all tuples u in the Branch relation, if the value of u on attribute bcity is Brooklyn, then the customer has an account at the branch whose name appears in the bname attribute of u." Division is difficult to understand. Think it through carefully.

A tuple relational calculus expression may generate an infinite expression, e.g.

{t | ¬ (t ∈ Borrow)}

There are an infinite number of tuples that are not in Borrow! Most of these tuples contain values that do not appear in the database. Therefore, we need to restrict the relational calculus a bit.

- The domain of a formula P, denoted dom(P), is the set of all values referenced in P.
- These include values mentioned in P as well as values that appear in a tuple of a relation mentioned in P.
- So, the domain of P is the set of all values explicitly appearing in P or that appear in relations mentioned in P.
- dom(t∈Borrow ∧ t[amount]>1200) is the set of all values appearing in Borrow and 1200.
- dom(t | ¬ (t∈Borrow) is the set of all values appearing in Borrow.

We may say an expression {t I P(t)} is safe if all values that appear in the result are values from dom(P). A safe expression yields a finite number of tuples as its result. Otherwise, it is called unsafe. The tuple relational calculus restricted to safe expressions is equivalent in expressive power to the relational algebra.

Summary

In the relational model, the data are stored in the form of tables or relations. Each table or relation has a unique name. Tables consist of a given number of columns or attributes. Every column of a table must have a name and no two columns of the same table may have identical names.

The rows of the table are called tuples. The total number of columns or attributes is known as the degree of the table. The relational model provides an alternative way to express queries. The principle of the relational model is set by E.F. Codd (Edgar Frank Codd) to use. This chapter also introduced concepts of relational algebra and relational calculus. different operators like selection, projection, rename, Cartesian product, union, intersection, natural join, division, and also assignment were discussed with suitable examples

Questions

1. What is the relational model?
2. What are the super key, candidate key, and primary key?
3. What are the differences between relational algebra and relational calculus?
4. What is the tuple?
5. What does it mean of word of relation in the relational model?
6. Please list down the kind of fundamental operation at least 6 operation.

Exercises

bname	loan#	cname	balance
ABA_TK	111	Gech	1000
ABA_KD	222	Kim	2000
SFU	333	Hayes	1500

bname	assets	bcity
ABA_TK	9,000,000	Takeo
ABA_KD	21,000,000	Kandal
SFU	17,000,000	PP

Figure 3.3 : The borrow and brach relations

According to figure 3.3 mentioned above, please write the formula of relational algebra to solve the problem below:

1. Show all tuples that have value “101” in column “loan#”
2. List all entries for ABA_TK (requires amount>2000)
3. Delete all of customer name “Gech” from branch name “ABA_TK”.
4. Delete all account at branches located in SFU.

Chapter 03

Relational Language-SQL

Objective

- To Understand history of Relational Language-SQL
- To explain DDL: Data Definition Language
- To understand Views
- To explain DML: Data Modification
- To understand Embedded SQL

In this chapter, you will learn:

3.1 History of SQL

SQL is the most popular computer language used to create, modify, retrieve and manipulate data from relational database management systems. According to history, during 1970s a group at IBM's San Jose research center developed a database and then Oracle to date. SQL is standardized by ANSI and ISO. The programming language is integrated with SQL or use SQL commands to use in database.As time goes on, SQL is more developed and more useful.

Commercial database systems require more user-friendly query languages. We will look at SQL in detail. Although referred to as query languages, SQL contains facilities for designing and modifying the database.

SQL (commonly expanded to Structured Query Language) is the most popular computer language used to create, modify, retrieve and manipulate data from relational database management systems (RDBMS). The language has evolved beyond its original purpose to support object-relational database management systems. It is an ANSI/ISO standard.

During the 1970s, a group at IBM's San Jose research center developed a database system "System R" based upon, but not strictly faithful to, Codd's model. Structured English Query Language ("SEQUEL") was designed to manipulate and retrieve data stored in System R. The acronym SEQUEL was later condensed to SQL because the word 'SEQUEL' was held as a trademark by the Hawker-Siddeley aircraft company of the UK. Although SQL was influenced by Codd's work, Donald D. Chamberlin and Raymond F. Boyce at IBM were the authors of the SEQUEL language design. Their concepts were published to increase interest in SQL.

The first non-commercial, relational, non-SQL database, Ingres, was developed in 1974 at U.C. Berkeley. In 1978, methodical testing commenced at customer test sites. Demonstrating both the usefulness and practicality of the system, this testing proved to be a success for IBM. As a result, IBM began to develop commercial products that implemented SQL based on their System R prototype, including the System/38 (announced in 1978 and commercially available in August 1979), SQL/DS (introduced in 1981), and DB2 (in 1983).

At the same time Relational Software, Inc. (now Oracle Corporation) saw the potential of the concepts described by Chamberlin and Boyce and developed their own version of a RDBMS for the Navy, CIA and others. In the summer of 1979 Relational Software, Inc. introduced Oracle V2 (Version2) for VAX computers as the first commercially available

implementation of SQL. Oracle is often incorrectly cited as beating IBM to market by two years, when in fact they only beat IBM's release of the System/38 by a few weeks. Considerable public interest then developed; soon many other vendors developed versions, and Oracle's future was ensured.

It is often suggested that IBM was slow to develop SQL and relational products, possibly because it wasn't available initially on the mainframe and Unix environments, and that they were afraid it would cut into lucrative sales of their IMS database product, which used navigational database models instead of relational.

But at the same time as Oracle was being developed, IBM was developing the System/38, which was intended to be the first relational database system, and was thought by some at the time, because of its advanced design and capabilities, that it might have become a possible replacement for the mainframe and Unix systems.

SQL was adopted as a standard by ANSI (American National Standards Institute) in 1986 and ISO (International Organization for Standardization) in 1987. ANSI has declared that the official pronunciation for SQL is 'Es-Kju-El', although many English-speaking database professionals still pronounce it as 'sequel'.

The SQL standard has gone through a number of revisions:

- 1986 SQL-86 (SQL-87): First published by ANSI. Ratified by ISO in 1987.
- 1989 SQL-89: Minor revision.
- 1992 SQL-92 (SQL2): Major revision.
- 1999 SQL: 1999 (SQL3) Added regular expression matching, recursive queries, triggers, non-scalar types and some object-oriented features. (The last two are somewhat controversial and not yet widely supported.)
- 2003 SQL: 2003: Introduced XML-related features, window functions, standardized sequences and columns with auto-generated values (including identity-columns).

Although SQL is defined by both ANSI and ISO, there are many extensions to and variations on the version of the language defined by these standards bodies. Many of these extensions are of a proprietary nature, such as Oracle Corporation's PL/SQL or Sybase, IBM's SQL PL (SQL Procedural Language) and Microsoft's Transact-SQL.

It is also not uncommon for commercial implementations to omit support for basic features of the standard, such as the DATE or TIME data types, preferring some variant of their own. As a result, in contrast to ANSI C or ANSI Fortran, which can usually be ported from platform to platform without major structural changes, SQL code can rarely be ported

between database systems without major modifications. There are several reasons for this lack of portability between database systems:

•the complexity and size of the SQL standard means that most databases do not implement the entire standard.

•the standard does not specify database behavior in several important areas (e.g. indexes) , leaving it up to implementations of the standard to decide how to behave.

• the SQL standard precisely specifies the syntax that a conformant database system must implement. However, the standard's specification of the semantics of language constructs is less well-de fined, leading to areas of ambiguity.

• many database vendors have large existing customer bases; where the SQL standard conflicts with the prior behavior of the vendor's database, the vendor may be unwilling to break backward compatibility.

• some believe the lack of compatibility between database systems is intentional in order to ensure vendor lock-in.

SQL is designed for a specific, limited purpose - querying data contained in a relational database. As such, it is a set-based, declarative computer language rather than an imperative language such as C or BASIC which, being programming languages, are designed to solve a much broader set of problems. Language extensions such as PL/SQL are designed to address this by turning SQL into a full- fledged programming language while maintaining the advantages of SQL.

Another approach is to allow programming language code to be embedded in and interact with the database. For example, Oracle and others include Java in the database, while PostgreSQL allows functions to be written in a wide variety of languages, including Perl, Tel, and C.

One joke about SQL is that "SQL is neither structured, nor is it limited to queries, nor is it a language." This is founded on the notion that pure SQL is not a classic programming language since it is not Turing-complete. On the other hand, however, it is a programming language because it has a grammar, syntax, and programmatic purpose and intent. The joke recalls Voltaire's remark that the Holy Roman Empire was "neither holy, nor Roman, nor an empire."

SQL contrasts with the more powerful database-oriented fourth-generation programming languages such as Focus or SAS in its relative functional simplicity and simpler command set. This greatly reduces the degree of difficulty involved in maintaining SQL source code,

but it also makes programming such questions as 'Who had the top ten scores?' more difficult, leading to the development of procedural extensions, discussed above.

However, it also makes it possible for SQL source code to be produced (and optimized) by software , leading to the development of a number of natural language database query languages, as well as 'drag and drop' database programming packages with 'object oriented' interfaces. Often these allow the resultant SQL source code to be examined, for educational purposes, further enhancement, or to be used in a different environment.

Technically, SQL is a declarative computer language for use with "SQL databases". Theorists and some practitioners note that many of the original SQL features were inspired by, but in violation of, the relational model for database management and its tuple calculus realization. Recent extensions to SQL achieved relational completeness, but have worsened the violations.

In addition, there are also some criticisms about the practical use of SQL:

- The language syntax is rather complex (sometimes called "COBOL-like").
- It does not provide a standard way, or at least a commonly-supported way, to split large commands into multiple smaller ones that reference each other by name. This tends to result in "run-on SQL sentences" and may force one into a deep hierarchical nesting when a graph-like (reference-by-name) approach may be more appropriate and better repetition-factoring.
- Implementations are inconsistent and, usually, incompatible between vendors.
- For larger statements, it is often difficult to factor repeated patterns and expressions into one or fewer places to avoid repetition and avoid having to make the same change to different places in a given statement.
- Confusion about the difference between value-to-column assignment in UPDATE and INSERT syntax.

Anyway, SQL has become the standard relational database language. It has several parts:

1. Data definition language (DDL) - provides commands to
   - Define relation schemes.
   - Delete relations.
   - Create indices.
   - Modify schemes.
2. Interactive data manipulation language (DML) - a query language based on both relational algebra and tuple relational calculus, plus commands to insert, delete and modify tuples.

3. View Definition - commands for defining views
4. Authorization - specifying access rights to relations and views.
5. Integrity - a limited form of integrity checking.
6. Transaction control - specifying beginning and end of transactions.
7. Embedded data manipulation language - for use within programming languages like C, PL/1, Cobol, Pascal, etc

We will only look at basic DDL, views and the DML. The relation schemes for the banking example used throughout the textbook are:

•Branch-scheme= (bname, bcity, assets)

Figure 4.1 the Branch relation

branch-name	*branch-city*	*assets*
Brighton	*Brooklyn*	*7100000*
Downtown	*Brooklyn*	*9000000*
Mianus	*Horseneck*	*400000*
North Town	*Rye*	*3700000*
Perryridge	*Horseneck*	*1700000*
Pownal	*Bennington*	*300000*
Redwood	*Palo Alto*	*2100000*
Round Hill	*Horseneck*	*8000000*

- Customer-scheme = (cname, street, ccity)

Figure 4.2 the Customer relation

customer-name	*customer-stree*	*customer-city*
Jones	*Main*	*Harrison*
Smith	*North*	*Rye*
Hayes	*Main*	*Harrison*
Curry	*North*	*Rye*
Lindsay	*Park*	*Pittsfield*
Turner	*Putnam*	*Stamford*
Williams	*Nassau*	*Princeton*
Adams	*Spring*	*Pittsfield*
Johnson	*Alma*	*Palo Alto*
Glenn	*Sand Hill*	*Woodside*
Brooks	*Senator*	*Brooklyn*
Green	*Walnut*	*Stamford*

- Depositor-scheme = (cname, account)

Figure 4.3 the Depositor relation

customer-name	*account-number*
Hayes	*A-102*
Johnson	*A-101*
Johnson	*A-201*
Jones	*A-217*
Lindsay	*A-222*
Smith	*A-215*
Turner	*A-305*

- Account-scheme = (account, bname, balance)

Figure 4.4 the Account relation

account-number	*branch-name*	*balance*
A-101	*Downtown*	*500*
A-102	*Perryridge*	*400*
A-201	*Brighton*	*900*
A-215	*Mianus*	*700*
A-217	*Brighton*	*750*
A-222	*Redwood*	*700*
A-305	*Round Hill*	*350*

- Loan-scheme = (loan, bname, amount)

Figure 4.5 the Loan relation

Loan	*branch-name*	*Amount*
L-11	*Round Hill*	*900*
L-14	*Downtown*	*1500*
L-15	*Perryridge*	*1500*
L-16	*Perryridge*	*1300*
L-17	*Downtown*	*1000*
L-23	*Redwood*	*2000*
L-93	*Mianus*	*500*

- Borrower-scheme = (cname, loan)

Figure 4.6 the Borrower relation

customer-name	*loan-number*
Adams	*L-16*
Curry	*L-93*
Hayes	*L-15*
Jackson	*L-14*
Jones	*L-17*
Smith	*L-11*
Smith	*L-23*
Willia.ms	*L-17*

3.2 DDL: Data Definition Language

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. In a SQL database, a schema is a list of logical structures of data. Attribute is or the conjunction of a column. Integrity Constraints are the protocols that a table's data columns must follow. Domain integrity constraint contains a certain set of rules or conditions to restrict the kind of attributes or values a column can hold in the database table. Referential Integrity is a constraint in the database that enforces the relationship between two tables Domain is value of attribute. Domain have type char(n) (or character(n)): fixed-length character string, with user-specified length. varchar(n) (or character varying): variable-length character string, with user-specified maximum length. int or integer: an integer (length is machine-dependent). MySQL primary key is a single or combination of the field, which is used to identify each record in a table uniquely.

The SQL DDL (Data Definition Language) allows specification of not only a set of relations, but also the following information for each relation:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints.
- The set of indices for each relation.
- Security and authorization information.
- Physical storage structure on disk.

3.2.1 Schema Definition in SQL

An SQL relation is defined by:

CREATE TABLE R (A_1 D_1, A_2 D_2, ... , A_n D_n,
INTETRITY-CONSTRAINT$_1$, ...
INTETRITY-CONSTRAINT$_N$)

where R is the relation name, Ai is the name of an attribute, and Di is the domain of that attribute.

The allowed integrity-constraints include

PRIMARY KEY (A_1, A_2, ..., A_N)
CHECK(P)

e.g.,

CREATE TABLE BRANCH (BNAME CHAR(l 5) NOT NULL
BCITY CHAR(30)
ASSETS INTEGER
PRIMARY KEY (BNAME)
CHECK (ASSETS>= 0))

The values of primary key must be ‘not null’ and ‘unique’. SQL-92 considers not null in PRIMARY KEY specification is redundant but SQL-89 requires to define it explicitly.

Check creates type checking functionality which could be quite useful.

e.g., CREATE TABLE STUDENT (NAME CHAR(15) NOT NULL
STUDENT-ID CHAR(10) NOT NULL
DEGREE-LEVEL CHAR(l 5) NOT NULL
CHECK (DEGREE-LEVEL IN ('BACHELORS', 'MASTERS',
'DOCTORATE')))

Some checking (such as foreign-key constraints) could be costly,

e.g., CHECK (BNAME IN (SELECT BNAME FROM BRANCH))

3.2.2 Integrity Constraints

Integrity constraints provide a way of ensuring that changes made to the database by authorized users do not result in a loss of data consistency. An integrity constraint can be any arbitrary predicate applied to the database. They may be costly to evaluate, so we will only consider integrity constraints that can be tested with minimal overhead.

An addition to the original standard allows specification of primary and candidate keys and foreign keys as part of the CREATE TABLE command:

- PRIMARY KEY clause includes a list of attributes forming the primary key.
- UNIQUE KEY clause includes a list of attributes forming a candidate key.
- FOREIGN KEY clause includes a list of attributes forming the foreign key, and the name of the relation referenced by the foreign key.

Domain Integrity Constraints

A domain of possible values should be associated with every attribute. These domain constraints are the most basic form of integrity constraint. They are easy to test for when data is entered.

Domain types

- Attributes may have the same domain, e.g. cname and employee-name.
- It is not as clear whether bname and cname domains ought to be distinct.
- At the implementation level, they are both character strings.
- At the conceptual level, we do not expect customers to have the same names as branches, in general.
- Strong typing of domains allows us to test for values inserted, and whether queries make sense.
- Newer systems, particularly object-oriented database systems, offer a rich set of domain types that can be extended easily.

The CHECK clause in SQL-92 permits domains to be restricted in powerful ways that most programming language type systems do not permit. The CHECK clause permits schema designer to specify a predicate that must be satisfied by any value assigned to a variable whose type is the domain.

e.g.,

```
CREATE DOMAIN HOURLY-WAGE NUMERIC(5, 2)
        CONSTRAINT WAGE-VALUE-TEST CHECK (VALUE >= 4.00)
```

Note that 'CONSTRAINT WAGE-VALUE-TEST’ is optional (to give a name to the test to signal which constraint is violated).

```
CREATE DOMAIN ACCOUNT-NUMBER CHAR(10)
                    CONSTRAINT ACCOUNT-NUMBER-NULL-TEST
                    CHECK(VALUE NOT NULL)

CREATE DOMAIN ACCOUNT-TYPE CHAR(10)
                    CONSTRAINT ACCOUNT-TYPE-TEST
                    CHECK(VALUE IN ('Checking', 'Saving'))
```

Referential Integrity Constraint

Often we wish to ensure that a value appearing in a relation for a given set of attributes also appears for another set of attributes in another relation. This is called *referential* integrity.

e.g.,

```
CREATE TABLE   CUSTOMER (CNAME CHAR(20) NOT NULL,
                    STREET CHAR(30),
                    CITY    CHAR(30),
                    PRIMARY KEY (CNAME))

CREATE TABLE   BRANCH (BNAME   CHAR(15) NOT NULL,
                    BCITY CHAR(30),
                    ASSETS  INTEGER,
                    PRIMARY KEY (BNAME),
                    CHECK (ASSETS > = 0))

CREATE TABLE    ACCOUNT (ACCOUNT# CHAR(10) NOT NULL ,
                    BNAME CHAR(l5),
                    BALANCE INTEGER,
                    PRIMARY KEY (ACCOUNT#),
                    FOREIGN KEY (BNAME) REFERENCES BRANCH,
                    CHECK (BALANCE>= 0) )
```

```
CREATE TABLE DEPOSITOR (CNAME CHAR(20) NOT NULL,
                        ACCOUNT# CHAR(10) NOT NULL,
                        PRIMARY KEY (CNAME, ACCOUNT#),
                        FOREIGN KEY (CNAME) REFERENCES CUSTOMER,
                        FOREIGN KEY (ACCOUNT#) REFERENCES ACCOUNT)
```

Notes on foreign keys: A short form to declare a single column is a foreign key.

```
BNAME CHAR(15) REFERENCES BRANCH
```

When a referential integrity constraint is violated, the normal procedure is to reject the action. But a FOREIGN KEY clause in SQL-92 can specify steps to be taken to change the tuples in the referenced relation to restore the constraint.

e.g.,

```
CREATE TABLE    ACCOUNT
        FOREIGN KEY (BNAME) REFERENCES BRANCH ,
        ON DELETE CASCADE
        ON INSERT CASCADE ,
```

If a delete of a tuple in Branch results in the preceding referential integrity constraints being violated, the delete is not rejected, but the delete 'cascade ' to the Account relation, deleting the tuple that refers to the Branch that was deleted. Update will be cascaded to the new value of the Branch!

SQL-92 also allows the FOREIGN KEY clause to specify actions other than cascade, such as setting the referencing field to NULL, or to a default value, if the constraint is violated.

If there is a chain of foreign key dependencies across multiple relations, a deletion or update at one end of the chain can propagate across the entire chain. If a cascading update or delete causes a constraint violation that cannot be handled by a further cascading operation, the system aborts the transaction and all the changes caused by the transaction and its cascading actions are undone.

Given and complexity and arbitrary nature of the way constraints in SQL behave with NULL values, it is the best to ensure that all columns of UNIQUE and FOREIGN KEY specifications are declared to be NONNULL.

3.2.3 Domain Types

The SQL-92 standard supports a variety of built-in domain types:

• CHAR (N) (or CHARACTER (N)): fixed-length character string, with user- specified length.

• VARCHAR (N) (or CHARACTER VARYING): variable-length character string,

• INT or INTEGER: an integer (length is machine-dependent).

• SMALLINT: a small integer (length is machine-dependent).

• NUMERIC (P, D): a fixed-point number with user-specified precision, consists of P digits (plus a sign) and D of P digits are to the right of the decimal point.

e.g., NUMERIC (3, 1) allows 44.5 to be stored exactly but not 444.5.

• REAL or DOUBLE PRECISION: floating-point or double-precision floating-point numbers, with machine-dependent precision.

• FLOAT (N): floating-point, with user-specified precision of at least N digits .

• DATE: a calendar date, containing four digit year, month, and day of the month.

• TIME: the time of the day in hours, minutes, and seconds.

SQL-92 allows arithmetic and comparison operations on various numeric domains, including, INTERVAL and cast (type coercion) such as transforming between SMALLINT and INT. It considers strings with different length are compatible types as well.

SQL-92 allows create domain statement,

e.g.,

CREATE DOMAIN PERSON-NAME CHAR(20)

3.3 Views

views is a virtual table based on the result-set of an SQL statement. index is an on-disk structure associated with a table or view that speeds retrieval of rows from the table or view.

3.3.1 View Definition

We have assumed up to now that the relations we are given are the actual relations stored in the database. For *security* and *convenience* reasons, we may wish to create a personalized collection of relations for a user.

We use the term view to refer to any relation, not part of the conceptual model that is made visible to the user as a *"virtual* relation". As relations may be modified by deletions, insertions and updates, it is generally not possible to store views. Views must then be recomputed for each query referring to them.

A view is defined using the CREATE VIEW command:

CREATE VIEW V AS <query expression>

where <query expression > is any legal query expression.

The view created is given the name V.

To create a view which contains all customers of all branches and their customers:

```
CREATE VIEW ALL-CUSTOMER AS
        (SELECT BNAME, CNAME FROM DEPOSITOR, ACCOUNT
        WHERE DEPOSTOR.ACCOUNT# = ACCOUNT.ACCOUNT#)
        UNION
        (SELECT BNAME, CNAME FROM BORROWER, LOAN
        WHERE BORROWER.LOAN#= LOAN.LOAN#)
```

Having defined a view, we can now use it to refer to the virtual relation it creates. View names can appear anywhere a relation name can.

e.g., We can now find all customers of the SFU branch by writing

```
SELECT CNAME FROM ALL-CUSTOMER WHERE BNAME=' SFU'
```

3.3.2 Updates Through Views and Null Values

Updates, insertions and deletions using views can cause problems. The modifications on a view must be transformed to modifications of the actual relations in the conceptual model of the database. The view update anomaly exists also in SQL. An example will illustrate: consider a clerk who needs to see all information in the loan relation except amount.

Let the view Branch-loan be given to the clerk:

```
CREATE VIEW BRANCH-LOAN AS (SELECT BNAME, LOAN#
                            FROM LOAN)
```

Since SQL allows a view name to appear anywhere a relation name may appear, the clerk can write:

INSERT INTO BRANCH-LOAN VALUES ('SFU', 'L-307')

This insertion is represented by an insertion into the actual relation LOAN, from which the view is constructed. However, we have no value for amount. This insertion results in 'SFU', 'L-307', NULL) being inserted into the LOAN relation. The symbol NULL says the value is unknown or does not exist.

As we saw, when a view is defined in terms of several relations, serious problems can result. As a result, many SQL-based systems impose the constraint that a modification is permitted through a view only if the view in question is defined in terms of *one* relation in the database.

3.3.3 Index Definition in SQL

Some SQL implementations include data definition commands to create and drop indices. The SQL commands are

An index is created by

create index <index -name>
on *r* (<attribute-list>)

The attribute list is the list of attributes in relation r that form the search key for the index.

.e.g., to create an index on bname for the branch relation:

create index *b-index*
on *branch (bname)*

If the search key is a candidate key, we add the word UNIQUE to the definition:

create unique index *b-index*
on *branch (bname)*

- If bname is not a candidate key, an error message will appear.
- If the index creation succeeds, any attempt to insert a tuple violating this requirement will

fail.

- The UNIQUE keyword is redundant if primary keys have been defined with integrity constraints already.

To remove an index, the command is

drop index <index-name>

3.4 DML: Data Modification

Interactive data manipulation language (DML) a query language based on both relational algebra and tuple relational calculus, plus commands to insert, delete and modify tuples. Data Modification (The INSERT, UPDATE, DELETE, SELECT and MERGE) statement are collectively referred to as DML (Data Manipulation Language) statements. Insert data into a relation, we either specify a tuple, or write a query whose result is the set of tuples to be inserted. Attribute values for inserted tuples must be members of the attribute's domain .Updating allows us to change some values in a tuple without necessarily changing all. Deletion is expressed in much the same way as a query. Instead of displaying, the selected tuples are removed from the database. We can only delete whole tuples. Data retrieval means obtaining data from a Database Management System (DBMS) such as ODBMS. Basic structure of an SQL expression consist of SELECT, FROM, and Where clause. The Rename Operation is mechanism to rename both relations and attributes. Tuple variable can be used in SQL, and are defined in the FROM clause. String is pattern matching using the operator LIKE. Patterns are case sensitive. Ordering the Display of Tuples. In SQL allow the user to control the order in which tuples are displayed (ORDER, DESC, ASC).Duplicate Tuples we use the DISTINCT Keyword. The SELECT DISTINCT statement is used to return only distinct (different) values. SQL has the set operations UNION, INTERSECT, EXCEPT. Aggregate function performs a calculation on a set of values, and returns a single value. SQL can then compute (average, minimum, maximum, total, number).Null value is a field with a NULL value is a field with no value. Netsted Subqueies is a SELECT query embedded within the WHERE or HAVING clause of another SQL query. We use the IN and NOT IN operations for set membership. To set compare set in terms of inequalities. The EXISTS operator is used to test for the existence of any record in a subquery. SQL-92 allows a subquery expression to be used in the from clause.Joined

Relations a logical connection that represents the relationship between two Cis. Join types: INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN).

3.4.1 Insert

To insert data into a relation, we either specify a tuple, or write a query whose result is the set of tuples to be inserted. Attribute values for inserted tuples must be members of the attribute's domain.

e.g.,

To insert a tuple for Smith who has $1200 in account A-9372 at the SFU branch.

```
INSERT INTO ACCOUNT VALUES ('SFU', 'A-9372', 1200)
```

To provide each loan that the customer has in the ‘SFU’ branch with a $200 savings account.

```
INSERT INTO ACCOUNT (SELECT BNAME, LOAN#, 200
                     FROM LOAN
                     WHERE BNAME='SFU')
```

Here, we use a SELECT to specify a set of tuples.

It is important that we evaluate the SELECT statement fully before carrying out any insertion. If some insertions were carried out even as the SELECT statement were being evaluated, the insertion

```
INSERT INTO ACCOUNT (SELECT *
                     FROM ACCOUNT)
```

might insert an infinite number of tuples. Evaluating the select statement *completely* before performing insertions avoids such problems.

It is possible for inserted tuples to be given values on only some attributes of the schema. The remaining attributes are assigned a NULL value denoted by NULL. We can prohibit the insertion of NULL values using the SQL DDL.

3.4.2 Update

Updating allows us to change some values in a tuple without necessarily changing all.

e.g.,

To increase all balances by 5 percent.

```
UPDATE ACCOUNT SET BALANCE=BALANCE*1.05
```

This statement is applied to every tuple in account.

To make two different rates of interest payment, depending on balance amount:

```
UPDATE ACCOUNT SET BALANCE=BALANCE * 1.06
                    WHERE BALANCE> 10000;
UPDATE ACCOUNT SET BALANCE=BALANCE * 1.05
                    WHERE BALANCE       10000
```

Note: in this example the order of the two operations is important.

In general, WHERE clause of UPDATE statement may contain any construct legal in a WHERE clause of a SELECT statement (including nesting).

A nested SELECT within an UPDATE may reference the relation that is being updated. As before, all tuples in the relation are first tested to see whether they should be updated, and the updates are carried out afterwards.

For example, to pay 5% interest on account whose balance is greater than average, we have

```
UPDATE ACCOUNT SET BALANCE=BALANCE * 1.05
          WHERE BALANCE> SELECT AVG (BALANCE)
                              FROM ACCOUNT
```

3.4.3 Delete

Deletion is expressed in much the same way as a query. Instead of displaying, the selected tuples are removed from the database. We can only delete whole tuples.

A deletion in SQL is of the form

```
DELETE FROM R WHERE P
```

Tuples in R for which P is true are deleted.

If the WHERE clause is omitted, *all* tuples are deleted. The request

```
DELETE FROM LOAN
```

deletes all tuples from the relation Loan.

Some more examples:

Delete all of Smith's account records.

```
DELETE FROM DEPOSITOR
        WHERE CNAME='Smith' .
```

Delete all loans with loan numbers between 1300 and 1500.

```
DELETE FROM LOAN
        WHERE LOAN# BETWEEN 1300 AND 1500
```

Delete all accounts at branches located in Surrey.

```
DELETE FROM ACCOUNT
        WHERE BNAME IN (SELECT BNAME
                        FROM BRANCH
                        WHERE BCITY=’Surrey’
```

We may only delete tuples from one relation at a time, but we may reference any number of relations in a select-from-where clause embedded in the WHERE clause of a DELETE.

However, if the DELETE request contains an embedded SELECT that references the relation from which tuples are to be deleted, ambiguities may result.

For example, to delete the records of all accounts with balances below the average, we might write

```
DELETE FROM ACCOUNT
    WHERE BALANCE< (SELECT AVG(BALANCE) FROM ACCOUNT)
```

You can see that as we delete tuples from account, the average balance changes.

Solution: The DELETE statement first test each tuple in the relation Account to check whether the account has a balance less than the average of the bank. Then all tuples that fail the test are deleted. Perform all the tests (and mark the tuples to be deleted) before any deletion then delete them en masse after the evaluations.

3.4.4 Select (Data Retrieval)

Basic structure of an SQL expression consists of SELECT, FROM and WHERE clauses:

- SELECT clause lists attributes to be copied - corresponds to relational algebra *project* (∏).
- FROM clause corresponds to *Cartesian product*(X) lists relations to be used.
- WHERE clause corresponds to *selection predicate* (σ) in relational algebra.

Typical query has the form

SELECT A_1, A_2, ….., A_n
FROM R_1, R_2, ….., R_m
WHERE *P*

where each Ai represents an attribute, each Ri a relation, and P is a predicate.

This is equivalent to the relational algebra expression

∏ A1, A2, ….., An {σ P (R1 X R2 X Rm)}

- If the WHERE clause is omitted, the predicate P is true.
- The list of attributes can be replaced with a * to select all.
- SQL forms the Cartesian product of the relations named, performs a selection using the predicate, then projects the result onto the attributes named.
- The result of an SQL query is a relation.
- SQL may internally convert into more efficient expressions.

e.g., Find the names of all branches in the Account relation.

```
SELECT BNAME
    FROM ACCOUNT
```

distinct vs. all: elimination or not elimination of duplicates.

Find the names of all branches in the Account relation.

```
SELECT DISTINCT BNAME
    FROM ACCOUNT
```

By default, duplicates are not removed. We can state it explicitly using all.

```
SELECT ALL BNAME
    FROM ACCOUNT
```

SELECT * means select all the attributes. Arithmetic operations can also be in the selection list.

The FROM clause by itself defines a Cartesian product of the relations in the clause. SQL does not have a natural join equivalent. However, natural join can be expressed in terms of a Cartesian product, selection, and projection.

For the relational algebra expression

$$\prod_{\text{cname, loan\#}} (\text{Borrow} \infty \text{Loan})$$

we can write in SQL,

```
SELECT DISTINCT CNAME, BORROWER.LOAN#
    FROM BORROWER, LOAN
    WHERE BORROWER.LOAN# = LOAN.LOAN#
```

More selections with join: "Find the names and loan numbers of all customers who have a loan at the SFU branch," we can write in SQL,

```
SELECT DISTINCT CNAME, BORROWER.LOAN#
```

```
FROM BORROWER, LOAN
WHERE BORROWER.LOAN#= LOAN.LOAN# and BNAME='SFU'
```

The predicates can be more complicated, and can involve

- Logical connectives such as ‘and, or, not’.
- Arithmetic expressions on constant or tuple values.
- The between operator for ranges of values.

e.g., Find account number of accounts with balances between $90,000 and $100,000.

```
SELECT ACCOUNT#
FROM ACCOUNT
WHERE BALANCE BETWEEN 90000 AND 100000
```

The Rename Operation

The rename operation is a mechanism to rename both relations and attributes.

AS clause can appear in both the SELECT and FROM clauses:

old-name AS new-name.

e.g.,

```
SELECT DISTINCT CNAME, BORROWER.LOAN# AS LOAN_ID
FROM BORROWER, LOAN
WHERE BORROWER.LOAN#= LOAN.LOAN# and BNAME= 'SFU'
```

Tuple Variables

Tuple variables can be used in SQL, and are defined in the FROM clause:

```
SELECT DISTINCT CNAME, T.LOAN#
FROM BORROWER AS S, LOAN AS T
WHERE S.LOAN# = T.LOAN#
```

Note: The keyword AS is optional here.

These variables can then be used throughout the expression. Think of it as being something like the rename operator.

Finds the names of all branches that have assets greater than at least one branch located in Burnaby.

```
SELECT DISTINCT T.BNAME
        FROM BRANCH S, BRANCH T
        WHERE S.BCITY=' Burnaby ' and T.ASSETS > S.ASSETS
```

String Operation

The most commonly used operation on strings is pattern matching using the operator LIKE. Patterns are case sensitive, e.g., 'Jim' does not match 'jim'.

String matching operators % (any substring) and _ (underscore, matching any character). E.g., '_ _ _ %' matches any string with at least 3 characters. We can use NOT LIKE for string mismatching.

e.g., Find all customers whose street includes the substring "Main".

```
SELECT CNAME
FROM CUSTOMER
WHERE STREET LIKE '%Main%'
```

Backslash overrides the special meaning of symbols. Use the keyword ESCAPE to define the escape character. E.g., LIKE 'ab%tely\%\' ESCAPE '\' matches all the strings beginning with 'ab' followed by a sequence of characters and then ' tely' and then '% \' .

SQL also permits a variety of functions on character strings, such as concatenating (using 'I'), extracting substrings, finding the length of strings, converting between upper case and lower case, and so on.

Ordering the Display of Tuples

SQL allows the user to control the order in which tuples are displayed.

- **ORDER** by makes tuples appear in sorted order (ascending order by default).
- **DESC** specifies descending order.
- **ASC** specifies ascending order.

```
SELECT *
        FROM LOAN
        ORDER BY AMOUNT DESC, LOAN# ASC
```

Sorting can be costly, and should only be done when needed.

Set Operations

SQL has the set operations UNION(U), INTERSECT(∩) and EXCEPT(−).

Find all customers having an account.

```
SELECT DISTINCT CNAME
        FROM DEPOSITOR
```

UNION: Find all customers having a loan, an account, or both.

```
(SELECT CNAME
        FROM DEPOSITOR)
UNION
(SELECT CNAME
        FROM BORROWER)
```

INTERSECT*: Find customers having a loan and an account.*

```
(SELECT CNAME
        FROM DEPOSITOR)
INTERSECT
(SELECT CNAME
        FROM BORROWER)
```

EXCEPT: Find customers having an account, but not a loan.

```
(SELECT CNAME
        FROM DEPOSITOR)
EXCEPT
(SELECT CNAME
        FROM BORROWER)
```

Some additional details:

• UNION eliminates duplicates, being a set operation. If we want to retain duplicates, we may use UNION ALL, similarly for INTERSECT and EXCEPT.

• Not all implementations of SQL have these set operations.

• EXCEPT in SQL-92 is called MINUS in SQL-86.

• It is possible to express these queries using other operations.

Aggregate Functions

In SQL we can compute functions on groups of tuples using the group by clause. Attributes given are used to form groups with the same values. SQL can then compute

- average value -- AVG
- minimum value -- MIN
- maximum value -- MAX
- total sum of values -- SUM
- number in group -- COUNT

These are called aggregate functions. They return a *single value.*

e.g., Find the average account balance at each branch.

```
SELECT BNAME, AVG (BALANCE)
        FROM CCOUNT
        GROUP BY BNAME
```

Find the number of depositors at each branch.

```
SELECT BNAME, COUNT (DISTINCT CNAME)
         FROM ACCOUNT, DEPOSITOR
        WHERE ACCOUNT.ACCOUNT#= DEPOSITOR.ACCOUNT#
        GROUP BY BNAME
```

We use DISTINCT so that a person having more than one account will not be counted more than once.

Find branches and their average balances where the average balance is more than $1200.

```
SELECT BNAME, AVG (BALANCE)
FROM ACCOUNT
GROUP BY BNAME
HAVING AVG (BALANCE)> 1200
```

Predicates in the HAYING clause are applied after the formation of groups.

Find the average balance of each customer who lives in ‘Vancouver’ and has at least three accounts:

```
SELECT DEPOSITOR.CNAME, AVG (BALANCE)
        FROM DEPOSITOR, ACCOUNT, CUSTOMER
        WHERE DEPOSITOR.CNAME= CUSTOMER.CNAME and
                ACCOUNT.ACCOUNT#= DEPOSITOR.ACCOUNT# and
        CCITY='VANCOUVER'
        GROUP BY DEPOSITOR.CNAME
        HAVING COUNT (DISTINCT ACCOUNT#) >= 3
```

If a WHERE clause and a HAYING clause appear in the same query, the WHERE clause predicate is applied first.

- Tuples satisfying WHERE clause are placed into groups by the GROUP BY clause.
- The HAVING clause is applied to each group.
- Groups satisfying the HAVING clause are used by the SELECT clause to generate the result tuples.
- If no HAVING clause is present, the tuples satisfying the WHERE clause are treated as a single group.

NULL Values

With insertions, we saw how NULL values might be needed if values were unknown. Queries involving NULLs pose problems.

If a value is not known, it cannot be compared or be used as part of an aggregate function. All comparisons involving NULL are *false* by definition. However, we can use the keyword NULL to test for NULL values:

```
SELECT DISTINCT LOAN#
    FROM LOAN
    WHERE AMOUNT IS NULL
```

All aggregate functions except COUNT ignore tuples with NULL values on the argument attributes.

3.4.5 Netsted Subqueies

Set Membership

We use the IN and NOT IN operations for set membership.

```
SELECT DISTINCT CNAME
        FROM BORROWER
        WHERE CNAME IN (SELECT CNAME
                                FROM ACCOUNT
                                WHERE BNAME='SFU')
```

Note that we can write the same query several ways in SQL.

We can also test for more than one attribute:

```
SELECT DISTINCT CNAME
    FROM BORROWER, LOAN
    WHERE BORROWER.LOAN#= LOAN.LOAN# and BNAME ='SFU'  and
    (BNAME, CNAME) IN (SELECT BNAME, CNAME
            FROM ACCOUNT, DEPOSITOR
            WHERE DEPOSITOR.ACCOUNT# = ACCOUNT.ACCOUNT#)
```

This finds all customers who have a loan and an account at the ‘SFU’ branch in yet another way. Finding all customers who have a loan but not an account, we can use the NOT IN operation.

Set Comparison

To compare set elements in terms of inequalities, we can write

```
SELECT DISTINCT T.BNAME
```

```
        FROM BRANCH T, BRANCH S
        WHERE T.ASSETS > S.ASSETS
              AND S.BCITY= 'BURNABY'
```

or we can write

```
SELECT BNAME
        FROM BRANCH
        WHERE ASSETS > SOME (SELECT ASSETS
                             FROM BRANCH
                             WHERE BCITY='BURNABY')
```

to find branches whose assets are greater than some branch in Burnaby.

We can use any of the equality or inequality operators with SOME. If we change > SOME to >ALL, we find branches whose assets are greater than all branches in Burnaby.

e.g., Find branches with the highest average balance. We cannot compose aggregate functions in SQL, e.g. we cannot do MAX(AVG(...)). Instead, we find the branches for which average balance is greater than or equal to all average balances:

```
SELECT BNAME
    FROM ACCOUNT
    GROUP BY BNAME
    HAVING AVG (BALANCE) >= ALL (SELECT AVG (BALANCE)
                                 FROM ACCOUNT
                                 GROUP BY BNAME)
```

Test for Empty Relations

The EXISTS construct returns true if the argument subquery is nonempty. Find all customers who have a loan and an account at the bank.

```
SELECT CNAME
        FROM BORROWER
        WHERE EXISTS (SELECT *
                      FROM DEPOSITOR
                      WHERE DEPOSITOR.CNAME =
                  BORROWER.CNAME)
```

Test for the Absence of Duplicate Tuples

The UNIQUE construct returns true if the argument subquery contains no duplicate tuples.

Find all customers who have only one account at the ‘SFU’ branch.

```
SELECT T.CNAME
        FROM DEPOSITOR AS T
        WHERE UNIQUE (SELECT R.CNAME
                      FROM ACCOUNT, DEPOSITOR AS R
                      WHERE T.CNAME = R.CNAME and
                       R.ACCOUNT# = ACCOUNT.ACCOUNT#
                      and ACCOUNT.BNAME = ' SFU' )
```

3.4.6 Joined Relations

Each variant of the join operations in SQL-92 consists of a join *type* and a join *condition.*

1.Join types: INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN

The keyword INNER and OUTER are optional since the rest of the join type enables us to deduce whether the join is an inner join or an outer join.

SQL-92 also provides two other join types:

- CROSS JOIN: an inner join without a join condition.
- UNION JOIN: a full outer join on the 'false' condition, i.e., where the inner join is empty.

2. Join conditions: NATURAL ON predicate USING (A_1, A_2, ..., A_N)

The use of join condition is mandatory for outer joins, but is optional for inner joins (if it is omitted, a Cartesian product results).

Two given relations: Loan and Borrower.

Figure 3.7 the Loan and Borrower relations

bname	*Loan#*	*amount*
Downtown	*L-170*	*3000*
Redwood	*L-230*	*4000*
Perryridge	*L-260*	*1700*

cname	*loan#*
Jones	*L-170*
Smith	*L-230*
Hayes	*L-155*

- Inner join:

LOAN INNER JOIN BORROWER ON LOAN.LOAN# = BORROWER.LOAN#

Notice that the loan# will appear twice in the inner joined relation.

Figure 3.8 the result of Loan inner join Borrower

bname	*loan#*	*amount*	*Cname*	*loan#*
Downtown	*L-170 L-*	*3000*	*Jones*	*1-170*
Redwood	*230*	*4000*	*Smith*	*1-230*

- Left outer join:

LOAN LEFT OUTER JOIN BORROWER ON LOAN.LOAN# = BORROWER.LOAN#

Figure 3.9 the result of Loan left outer join Borrower

bname	*loan#*	*amount*	*cname*	*loan#*
Downtown	*L-170*	*3000*	*Jones*	*L-170*
Redwood	*L-230*	*4000*	*Smith*	*L-230*
Perryridge	*L-280*	*1700*	*nu11*	*null*

- Natural inner join: _

LOAN NATURAL INNER JOIN BORROWER

Figure 3.10 the result of Loan natural inner join Borrower

bname	*loan#*	*amount*	*cname*
Downtown	*L-170*	*3000*	*Jones*
Redwood	*L-230*	*4000*	*Smith*

- Natural full outer join :

LOAN NATURAL FULL OUTER JOIN BORROWER USING(LOAN#)

Figure 3.11 the result of Loan natural full outer join Borrower using (loan#)

bname	*loan#*	*amount*	*cname*
Downtown	*L-170*	*3000*	*Jones*
Redwood	*L-230*	*4000*	*Smith*
Perryridgc	*L-260*	*1700*	*null*
null	*L-155*	*null*	*Hayes*

Find all customers who have either an account or a loan (but not both) at the bank.

```
SELECT CNAME
        FROM (NATURAL FULL OUTER JOIN BORROWER)
        WHERE ACCOUNT# IS NULL OR LOAN# IS NULL
```

3.5 Embedded SQL

Other SQL Features is special language to assist application programmers in creating temples on the screen for a user interface.

SQL provides a powerful declarative query language. However, access to a database from a general-purpose programming language is required because,

- SQL is not as powerful as a general-purpose programming language. There are queries that cannot be expressed in SQL, but can be programmed in C, Java, etc.
- Nondeclarative actions - such as printing a report, interacting with a user, or sending the result to a GUI - cannot be done from within SQL.

The SQL standard defines embedding of SQL as embedded SQL and the language in which SQL queries are embedded is referred as *host* language. The result of the query is made available to the program one tuple (record) at a time.

To identify embedded SQL requests to the preprocessor, we use EXEC SQL statement:

```
EXEC SQL embedded_SQL_statement END-EXEC
```

Embedded SQL statements: DEC LARE CURSOR, OPEN, and FETCH statements.

```
EXEC SQL
        DECLARE C CURSOR FOR
                SELECT CNAME, CCITY
                        FROM DEPOSIT, CUSTOMER
```

```
                    WHERE DEPOSIT.CNAME = CUSTOMER.CNAME
                        and DEPOSIT.BALANCE> :AMOUNT
END-EXEC
```

where: AMOUNT is a host-language variable.

```
EXEC SQL
        OPEN C
END-EXEC
```

This statement causes the DB system to execute the query and to save the results within a temporary relation.
A series of FETCH statement are executed to make tuples of the results available to the program.

```
EXEC SQL
        FETCH C INTO :CN, :CC
END-EXEC
```

The program can then manipulate the variable: CN and :cc using the features of the host programming language.
A single FETCH request returns only one tuple. We need to use a WHILE loop (or equivalent) to process each tuple of the result until no further tuples (when a variable in the SQLCA is set).
We need to use close statement to tell the DB system to delete the temporary relation that held the result of the query.

```
EXEC SQL
        CLOSEC
END-EXEC
```

Embedded SQL can execute any valid UPDATE, INSERT, or DELETE statements. Dynamic SQL component allows programs to construct and submit SQL queries at run

time. SQL-92 also contains a module language, while allows procedures to be defined in SQL.

Summary

- *SQL is the most popular computer language used to create, modify, retrieve and manipulate data from relational database management systems. According to history, during 1970s a group at IBM's San Jose research center developed a database and then Oracle to date. SQL is standardized by ANSI and ISO. The programming language is integrated with SQL or use SQL commands to use in database.As time goes on, SQL is more developed and more useful.*
- *DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema.*
- *In a SQL database, a schema is a list of logical structures of data. Attribute is or the conjunction of a column.*
- *Integrity Constraints are the protocols that a table's data columns must follow.*
- *Domain integrity constraint contains a certain set of rules or conditions to restrict the kind of attributes or values a column can hold in the database table.*
- *Referential Integrity is a constraint in the database that enforces the relationship between two tables.*
- *Domain is value of attribute. Domain have type char(n) (or character(n)): fixed-length character string, with user-specified length. varchar(n) (or character varying): variable-length character string, with user-specified maximum length. int or integer: an integer (length is machine-dependent). MySQL primary key is a single or combination of the field, which is used to identify each record in a table uniquely.*
- *views is a virtual table based on the result-set of an SQL statement.*
- *index is an on-disk structure associated with a table or view that speeds retrieval of rows from the table or view.*
- *Interactive data manipulation language (DML) a query language based on both relational algebra and tuple relational calculus, plus commands to insert, delete and modify tuples.*

- *Data Modification (The INSERT, UPDATE, DELETE, SELECT and MERGE) statement are collectively referred to as DML (Data Manipulation Language) statements.*
- *Insert data into a relation, we either specify a tuple, or write a query whose result is the set of tuples to be inserted. Attribute values for inserted tuples must be members of the attribute's domain.*
- *Updating allows us to change some values in a tuple without necessarily changing all.*
- *Deletion is expressed in much the same way as a query. Instead of displaying, the selected tuples are removed from the database. We can only delete whole tuples.*
- *Data retrieval means obtaining data from a Database Management System (DBMS) such as ODBMS. Basic structure of an SQL expression consist of SELECT, FROM, and Where clause.*
- *The Rename Operation is mechanism to rename both relations and attributes.*
- *Tuple variable can be used in SQL, and are defined in the FROM clause.*
- *String is pattern matching using the operator LIKE. Patterns are case sensitive.*
- *Ordering the Display of Tuples. In SQL allow the user to control the order in which tuples are displayed (ORDER, DESC, ASC).*
- *Duplicate Tuples we use the DISTINCT Keyword. The SELECT DISTINCT statement is used to return only distinct (different) values.*
- *SQL has the set operations UNION, INTERSECT, EXCEPT.*
- *Aggregate function performs a calculation on a set of values, and returns a single value. SQL can then compute (average, minimum, maximum, total, number).*
- *Null value is a field with a NULL value is a field with no value.*

- *Netsted Subqueies is a SELECT query embedded within the WHERE or HAVING clause of another SQL query.*
- *We use the IN and NOT IN operations for set membership.*
- *To set compare set in terms of inequalities. The EXISTS operator is used to test for the existence of any record in a subquery. SQL-92 allows a subquery expression to be used in the from clause.*
- *Joined Relations a logical connection that represents the relationship between two Cis. Join types: INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN).*
- *Other SQL Features is special language to assist application programmers in creating temples on the screen for a user interface.*

Questions

1. What is DDL: Data Definition Language?
2. What are type of constraint in SQL?
3. What is Domain type? Please detail each section of the Domain Type?
4. What is view?
5. What is DML: Data Modification?
6. What is Insert? Please give command example?
7. What is Update? Please give command example?
8. What is Delete? Please give command example?
9. What is Select (Data Retrieval)? Please give command example?
10. How do you order a displayed of tuples?
11. What is aggregate functions in SQL with example?
12. What are operations for use in set membership?
13. What are comparison operator used in conditions that compares one expression with another?
14. What is Jointed Relation?
15. What are the Type of Join in SQL?
16. What is embedded SQL? Please give example?

Exercises

1. What is the syntax to create a database in SQL?
2. What is the syntax to create a Table in SQL?
3. What is the syntax for insert in SQL?
4. What is the syntax of update in SQL?
5. What is the syntax of Select command in SQL?
6. What is the syntax of the aggregate function in SQL?
7. What is the syntax of ORDER BY in SQL?
8. What is the syntax of join in SQL?

Chapter 04

Entity-Relationship Model

Objective

- To understand Entity-Relationship model
- To understand Entity and Entity Set
- To explain Relationship and Relationship Set
- To explain Keys
- To explain E-R Diagram
- To understand Reducing E-R Diagrams to Tables

In this chapter, you will learn:

4.1 Overview

Data modeling is the process of analyzing and defining all the different data your business collects and produces, as well as the relationships between those bits of data. The data modeling the entity-relational mode is intended primarily for the DB design and the entity-relationship model or entity-relationship diagram (ERD) is a data model or diagram for high-level description of conceptual data models.

The entity-relationship model or entity-relationship diagram (ERD) is a data model or diagram for high-level descriptions of *conceptual data models*, and it provides a *graphical notation* for representing such data models in the form of entity relationship diagrams. Such models are typically used in the first stage of information-system design; they are used, for example, to describe information needs and/or the type of information that is to be stored in the database during the requirements analysis.

The data modeling technique, however, can be used to describe any ontology (i.e.an overview and classifications of used terms and their relationships) for a certain universe of discourse (i.e. area of interest). In the case of the design of an information system that is based on a database, the conceptual data model is, at a later stage (usually called logical design), mapped to a logical data model, such as the relational model; this in tum is mapped to a physical model during physical design. Note that sometimes, both of these phases are referred to as "physical design". The E-R (entity-relationship) data model views the real world as a set of basic objects (entities) and relationships among these objects.

The entity-relationship mode is intended primarily for the DB design process by allowing the specification of an enterprise scheme. This represents the overall logical structure of the DB.

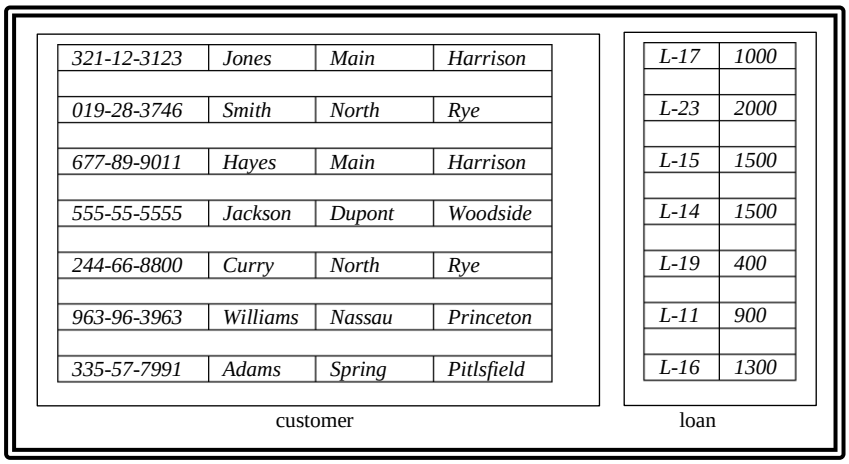
4.2 Entity and Entity Set

An entity is a real-world thing or a real-world object which is distinguishable from other objects in the real world. An entity set is a set of entities of the same type. Attribute is a function which maps an entity set into a domain. The domain is the set of permitted values.

An *entity* is an object that exists and is distinguishable from other objects. For instance, "John Harris with S.S.N. (Social Security Number) 890-12-3456" is an entity, as he can be uniquely identified as one particular person in the universe. An entity may be concrete (a person or a book, for example) or abstract (like a holiday or a concept).

An *entity set* is a set of entities of the same type (e.g., all persons having an account at a bank). Entity sets need not be disjoint. For example, the entity set "employee" (all employees of a bank) and the entity set "customer" (all customers of the bank) may have members in common. Figure 4.1 shows two entity sets, "customer" and "loan".

Figure 4.1 two entity sets, "customer" and "loan"



Attributes

An entity is represented by a set of *attributes*, e.g. name, S.S.N., street, city for "customer" entity. The *domain* of the attribute is the set of permitted values (e.g. the telephone number must be seven positive integers).

Formally, an attribute is a function which maps an entity set into a domain. Every entity is described by a set of (attribute, data value) pairs. There is one pair for each attribute of the entity set. E.g. a particular "customer" entity is described by the set {(name, Harris), (S.S.N., 890-123-456), (street, North), (city, Georgetown)}.

An analogy can be made with the programming language notion of type definition. The concept of an entity set corresponds to the programming language type definition. A variable of a given type has a particular value at a point in time. Thus, a programming language variable corresponds to an entity in the E-R model.

For example, five entity sets and their attributes are introduced below:

• Branch, the set of all branches of a particular bank. Each branch is described by the attributes branch-name, branch-city and assets.

• Customer, the set of all people having an account at the bank. Attributes are customer-name, S.S.N., street and customer-city.

• Employee, with attributes employee-name and phone-number.

• Account, the set of all accounts created and maintained in the bank. Attributes are account -number and balance.

• Transaction, the set of all account transactions executed in the bank. Attributes are transaction-number, date and amount.

4.3 Relationship and Relationship Set

A relationship is an association among several entities. A relationship set is a collection of relationships of the same type, and an entity set is a collection of entities of the same type. The terms superkey, candidate key, and primary key apply to entity and relation-ship sets as they do for relation schemas. Identifying the primary key of a relationship set requires some care, since it is composed of attributes from one or more of the related entity sets. Mapping cardinalities express the number of entities to which another entity can be associated via a relationship set. There are 4 mapping cardinality must be one of: one-to-one.one-to-many, many-to-one and many-to-many.

A relationship is an *association* between several entities. A relationship set is a set of relationships of the same type.

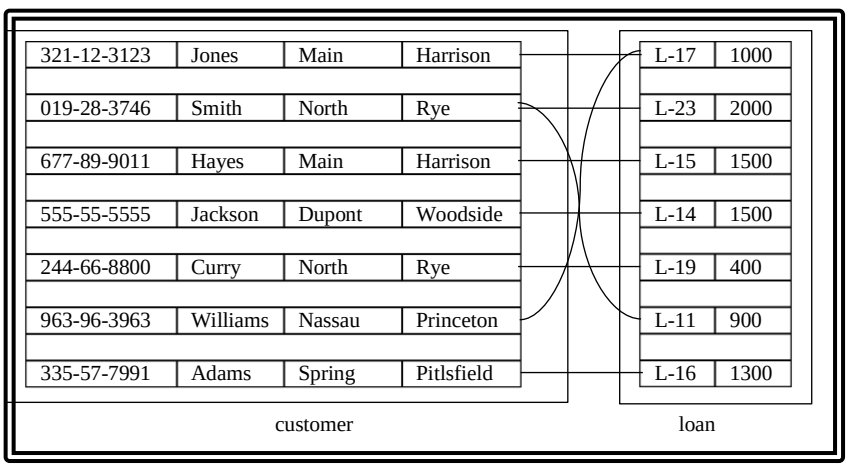
Formally it is a mathematical relation on ≥ 2 (possibly non-distinct) sets. If E_1, E_2,…, E_n are entity sets, then a relationship set R is a *subset* of

$\{(e_1, e_2,\ldots,e_n)|\ e_1 \in E_1, e_2 \in E_2, \ldots , e_n \in E_n\}$

, where (e_1 , e_2, ... ,e_n) is a relationship.

For example, consider the two entity sets customer and loan shown in Fig. 4.1. We define the relationship CustLoan to denote the association between customers and their loans. This is a binary relationship set shown in Figure 4.2.

Figure 4.2 a binary relationship set between customers and their loans



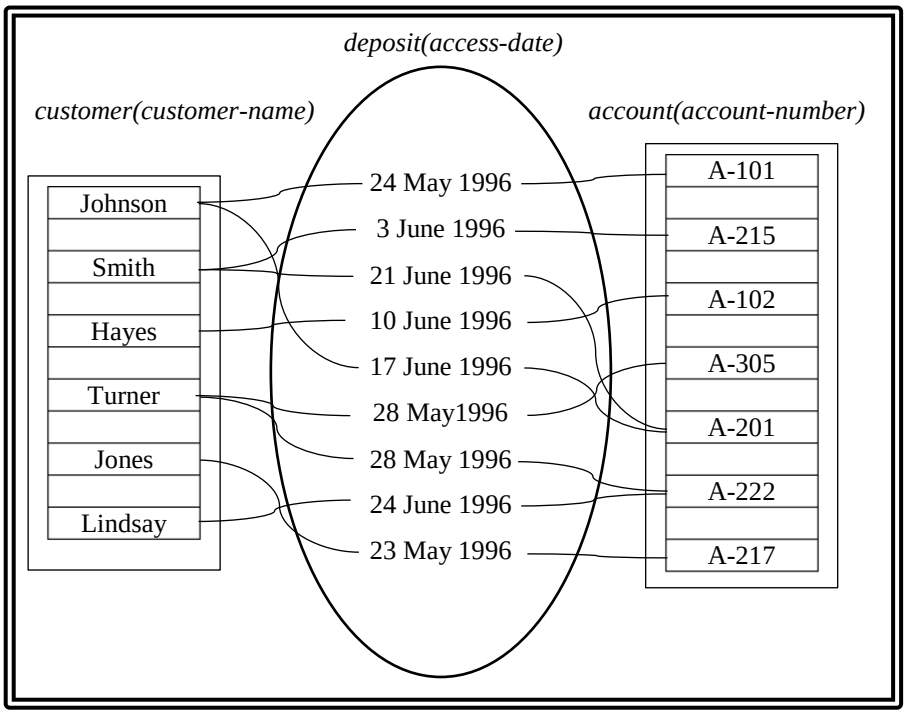
Going back to the formal definition, the relationship set CustLoan is a subset of all the possible customer and account pairings. This is a binary relationship. Occasionally there are relationships involving more than two entity sets.

Roles and Attributes

The *role* of an entity is the function it plays in a relationship. For example, the relationship works-for could be ordered pairs of employee entities. The first employee takes the role of manager, and the second one will take the role of worker.

A relationship may also have descriptive attributes. For example, date (last date of account access) could be an attribute of the depositor relationship set between the customer and account entity set shown in Figure 4.3.

Figure 2.3 the depositor relationship set



It is possible to define a set of entities and the relationships among them in a number of different ways. The main difference is in how we deal with attributes.

Consider the entity set employee with attributes employee-name and phone- number. We could argue that the phone be treated as an entity itself, with attributes phone-number and location. Then we have two entity sets, and the relationship set EmpPhn defining the association between employees and their phones. This new definition allows employees to have several (or zero) phones. New definition may more accurately reflect the real world.

We cannot extend this argument easily to making employee-name an entity. The question of what constitutes an entity and what constitutes an attribute depends mainly on the structure of the real world situation being modeled, and the semantics associated with the attribute in question.

An E-R scheme may define certain constraints to which the contents of a database must conform: *mapping cardinalities* and *existence dependencies.*

Mapping cardinalities

Mapping cardinalities express the number of entities to which another entity can be associated via a relationship. For binary relationship sets between entity sets A and B, the mapping cardinality must be one of:

• one-to-one: An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A. (Figure 4.4 (a))

• one-to-many: An entity in A is associated with any number in B. An entity in B is associated with at most one entity in A. (Figure 4.4(b))

• many-to-one: An entity in A is associated with at most one entity in B. An entity in B is associated with any number in A. (Figure 4.5 (a))

• many-to-many: Entities in A and B are associated with any number from each other. (Figure 4.5(b))

Figure 4.4 (a) one-to-one (b) one-to-many

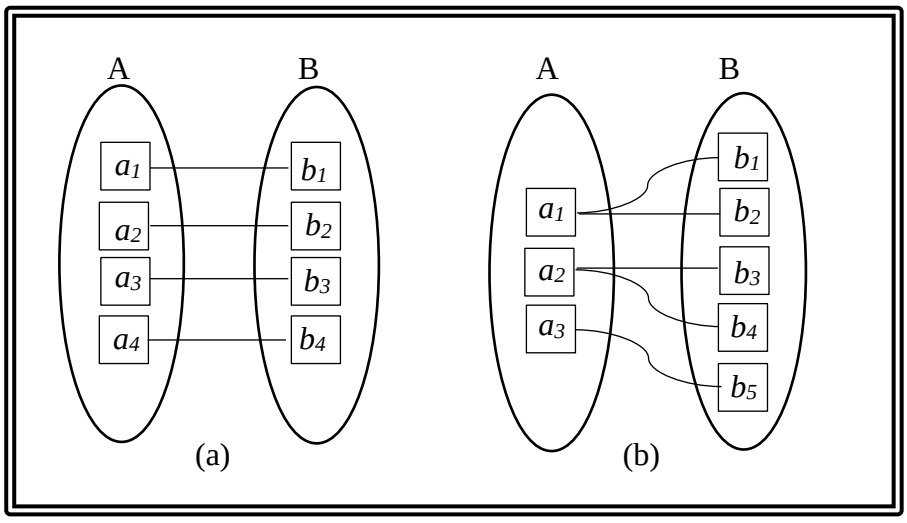
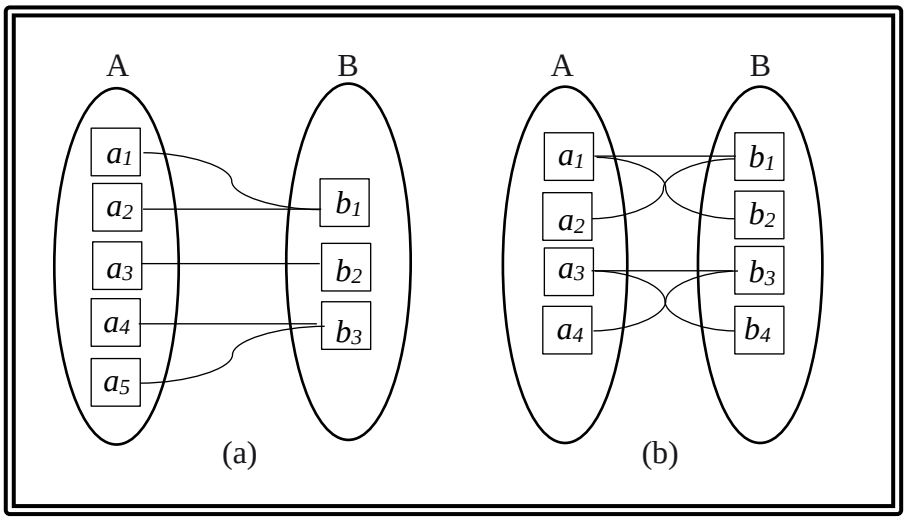


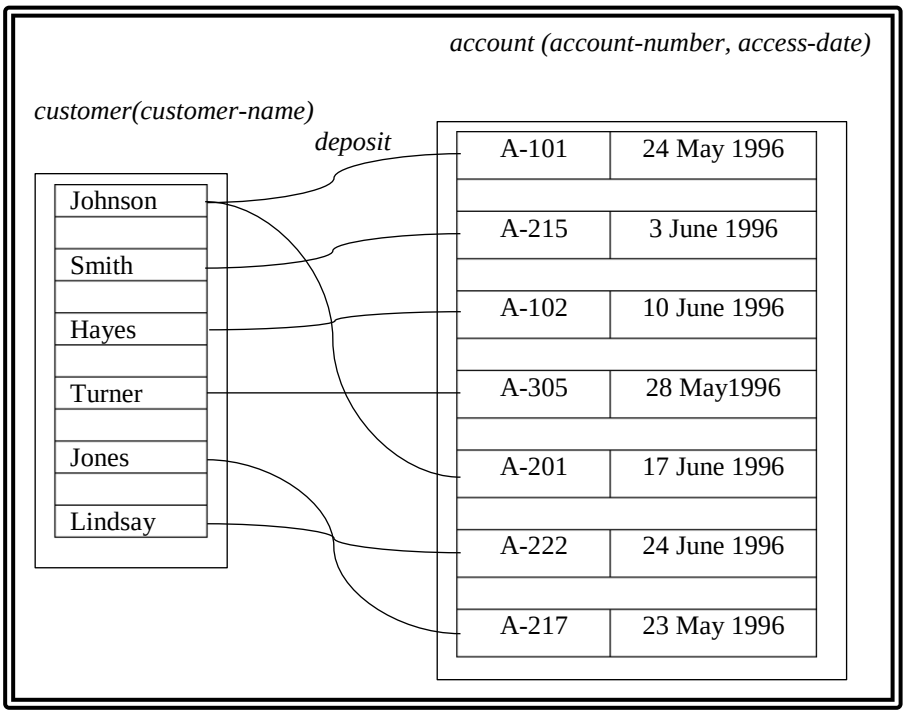
Figure 4.5 (a) many-to-one (b) many-to-many



The appropriate mapping cardinality for a particular relationship set depends on the real world being modeled. (Think about the CustLoan relationship in Figure 4.2.) Further, mapping cardinalities affect the ER design.

For example, we can make access-date an *attribute* of account in Figure 4.4, instead of a relationship attribute, 1 each account can have only one customer. I.e., the relationship from account to customer is many to one, or equivalently, customer to account is one to many as shown in Figure 4.6.

Figure 4.6 the relationship from account to customer



Existence dependencies

Existence dependencies are that if the existence of entity X depends on the existence of entity Y, then X is said to be existence dependent on Y. (Or we say that Y is the dominant entity and X is the subordinate entity.)

For example, consider account and transaction entity sets, and a relationship log between them. This is one-to-many from account to transaction. If an account entity is deleted, its associated transaction entities must also be deleted. Thus account is dominant and transaction is subordinate.

4.4 Keys

Key is an attribute or a set of attributes that help to uniquely identify a tuple (or row) in a relation (or table) type of Key in DB: Primary Key, Super Key. Candidate Key, Alternate Key, Foreign Key, Composite Key, Unique Key. The terms superkey, candidate key, and primary key apply to entity and relationship sets as they do for relation schemas. Identifying the primary key of a relationship set requires some care, since it is composed of attributes from one or more of the related entity sets.

Differences between entities must be expressed in terms of attributes. A *superkey* is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set. For example, in the entity set customer, customer-name and S.S.N. is a superkey. Note that customer-name alone is not, as two customers could have the same name.

A superkey may contain extraneous attributes, and we are often interested in the smallest superkey. A superkey for which no subset is a superkey is called a *candidate key*. In the example above, S.S.N. is a candidate key, as it is minimal, and uniquely identifies a customer entity.

A *primary key* is a candidate key (there may be more than one) chosen by the DB designer to identify entities in an entity set. An entity set that does not possess sufficient attributes to form a primary key is called a weak entity set. One that does have a primary key is called a strong entity set.

Primary keys for weak entity sets

For example, the entity set transaction has attributes transaction-number, date and amount. Different transactions on different accounts could share the same number. These are not sufficient to form a primary key (uniquely identify a transaction). Thus transaction is a weak entity set.

For a weak entity set to be meaningful, it must be part of a one-to-many relationship set. This relationship set should have no descriptive attributes. (Why?) The idea of strong and weak entity sets is related to the existence dependencies seen earlier.

Member of a strong entity set is a dominant entity. Member of a weak entity set is a subordinate entity. A weak entity set does not have a primary key, but we need a means of distinguishing among the entities. The *discriminator* of a weak entity set is a set of attributes that allows this distinction to be made. The primary key of a weak entity set is formed by taking the primary key of the strong entity set on which its existence depends (see Mapping Constraints) plus its discriminator.

To illustrate: transaction is a weak entity. It is existence-dependent on account. The primary key of account is account-number. Transaction-number distinguishes transaction entities within the same account (and is thus the discriminator). So the primary key for transaction would be (account-number, transaction-number).

Just Remember: the primary key of a weak entity is found by taking the primary key of the strong entity on which it is existence-dependent, plus the discriminator of the weak entity set.

Primary keys for relationship sets

The attributes of a relationship set are the attributes that comprise the primary keys of the entity sets involved in the relationship set. For example, S.S.N. is the primary key of customer, and account-number is the primary key of account. The attributes of the relationship set CustAcct are then (account-number, S.S.N.).

This is enough information to enable us to relate an account to a person. If the relationship has descriptive attributes, those are also included in its attribute set. For example, we might add the attribute date to the above relationship set, signifying the date of last access to an account by a particular customer. Note that this attribute cannot instead be placed in either entity set as it relates to both a customer and an account, and the relationship is many-to-many.

The primary key of a relationship set R depends on the mapping cardinality and the presence of descriptive attributes. With no descriptive attributes:

1. many-to-many: all attributes in R.
2. one-to-many: primary key for the "many" entity.
3. one-to-one: primary key of either entity.

Descriptive attributes may be added, depending on the mapping cardinality and the semantics involved.

4.5 E-R Diagram

A database design specified by an E-R diagram can be represented by a collection of relation schemas. For each entity set and for each relationship set in the database, there is a unique relation schema that is assigned the name of the corresponding entity set or relationship set. This forms the basis for deriving a relational database design from an E-R diagram. E-R diagram basic components are: rectangles ellipses, diamonds and lines.

There are a number of conventions for entity-relationship diagrams (ERDs). The classical notation is described in the remainder of this article, and mainly relates to conceptual modeling. There are a range of notations more typically employed in logical and physical database design, including information engineering, IDEF1x (ICAM DEFinition Language) and dimensional modeling.

We can express the overall logical structure of a database graphically with an E- R diagram. Its basic components are:

1. rectangles representing entity sets.
2. ellipses representing attributes.
3. diamonds representing relationship sets.
4. lines linking attributes to entity sets and entity sets to relationship sets. In the text, lines may be directed (have an arrow on the end) to signify mapping cardinalities for relationship sets.

The summaries of symbols used in E-R notation are shown in Figures 4.7, 4.8.

Figure 4.7 the summary of symbols

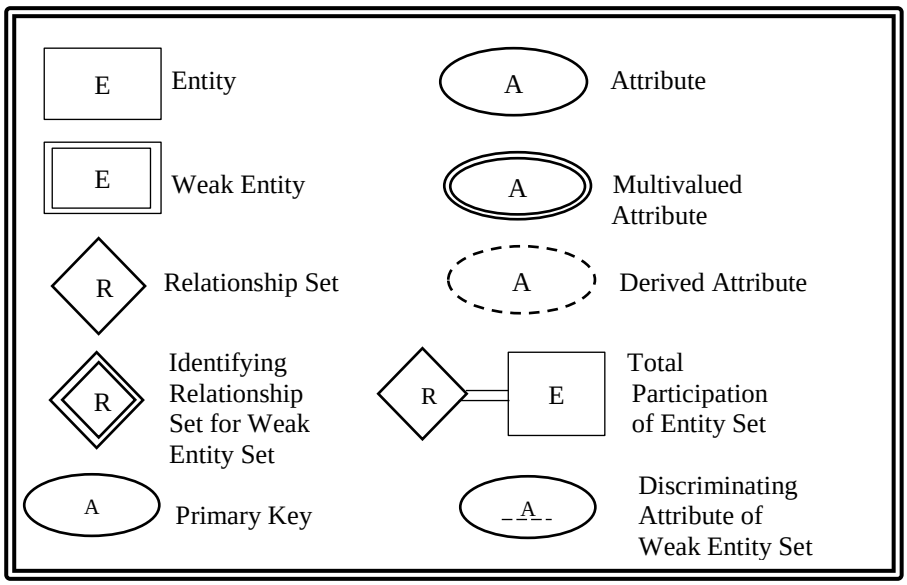


Figure 4.8 the summary of symbols

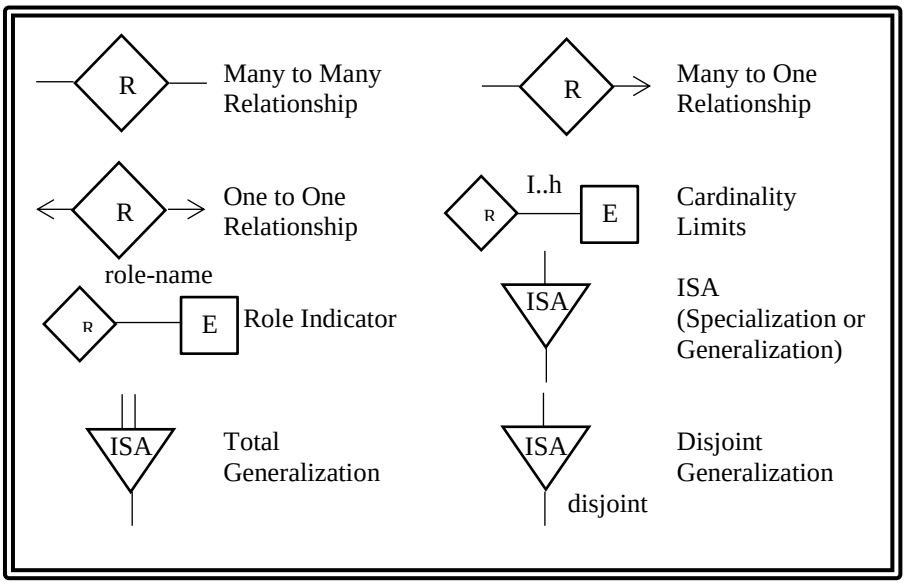
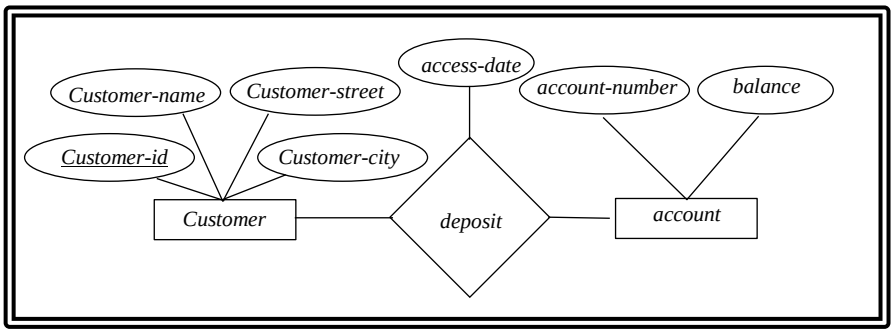


Figure 4.9 shows an example with two entity sets and a relationship set. In this example, the primary key of customer is customer-id, and the primary key of account is account-number respectively. Go back and review mapping cardinalities. They express the number of entities to which an entity can be associated via a relationship. The arrow positioning is simple once you get it straight in your mind. Think of the arrow head as pointing to the entity that "one" refers to. Thus, in this example, the cardinality of the relationship set depositor is many-to-many.

Figure 4.9 an example with two entity sets and a relationship set



Other styles of E-R diagram

The text uses one particular style of diagram. Many variations exist. Some of the variations we will see are:

• Diamonds being omitted - a link between entities indicates a relationship. (See Figure 4.10)

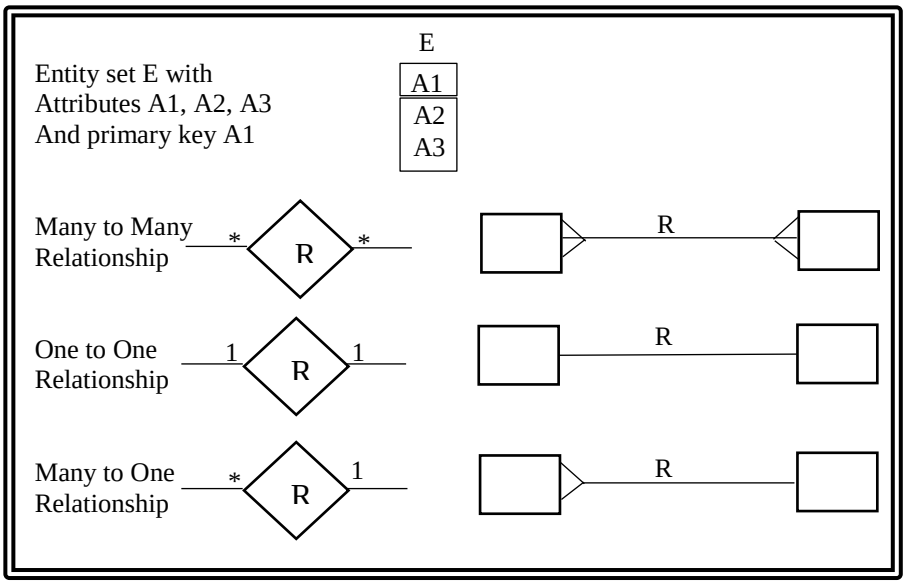
- Less symbols, clearer picture.
- What happens with descriptive attributes?

- In this case, we have to create an intersection entity to possess the attributes.

• Numbers instead of arrowheads indicating cardinality. (See Figure 4.10)

- Symbols, 1, n and m used.
- E.g. 1 to 1, 1 to n, n to m.
- Easier to understand than arrowheads.

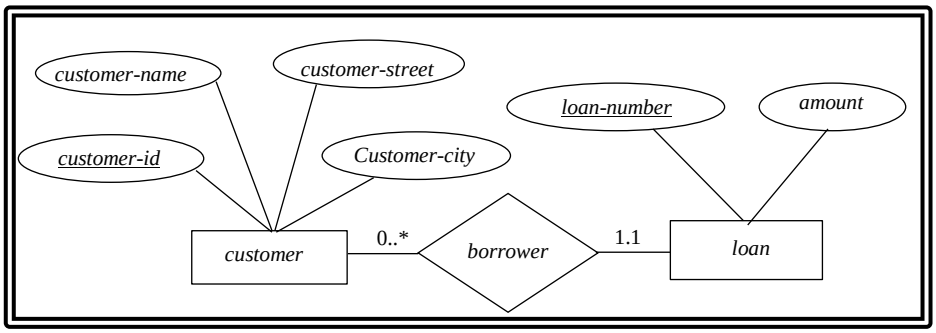
Figure 4.10 numbers instead of arrowheads indicating cardinality



• A range of numbers indicating optionality of relationship. (See Figure 4.11)

- E.g (0,1) indicates minimum zero (optional), maximum 1.
- Can also use (0,n), (1,1) or (1 ,n).
- Typically used on near end of link - confusing at first, but gives more information.
- E.g. entity 1 (0, 1) -- (1,n) entity 2 indicates that entity 1 is related to between 0 and 1 occurrences of entity 2 (optional).
- Entity 2 is related to at least 1 and possibly many occurrences of entity 1 (mandatory).

Figure 4.11 optionality of relationship



• Multivalued attributes may be indicated in some manner.

- Means attribute can have more than one value, e.g. hobbies.
- Has to be normalized later on.

• Extended E-R diagrams allowing more details/constraints in the real world to be recorded. (See Figures 4.12, 4,13)

- Composite attributes.
- Derived attributes.
- Subclasses and super classes.
- Generalization and specialization.

Figure 4.12 composite attributes

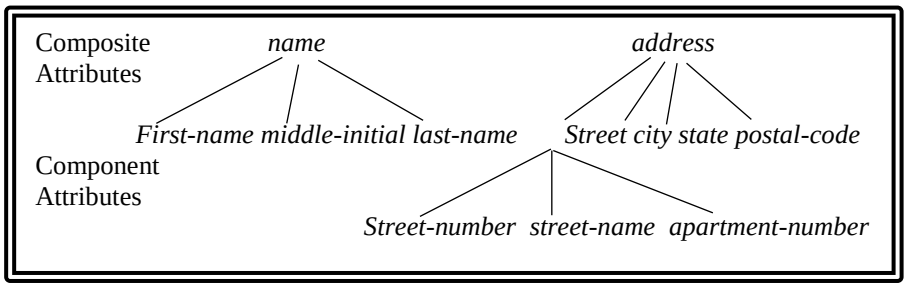
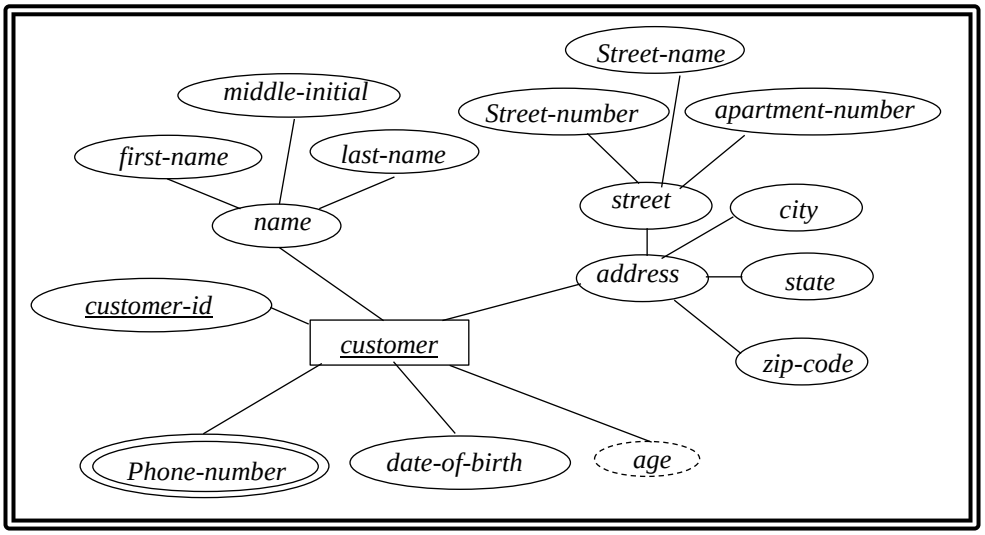


Figure 4.13 multivalue and derived attributes

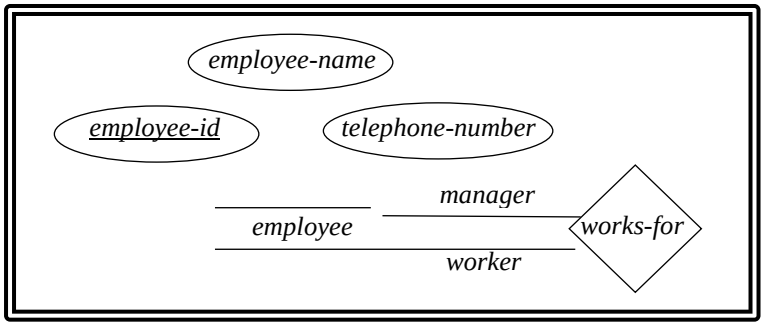


Roles in E-R Diagrams

The function that an entity plays in a relationship is called its role. Roles are normally explicit and not specified. They are useful when the *meaning* of a relationship set needs clarification. For example, the entity sets of a relationship may not be distinct. The relationship works-for might be ordered pairs of employees (first is manager, second is worker).

In the E-R diagram, this can be shown by labeling the lines connecting entities (rectangles) to relationships (diamonds). (See figure 9.14).

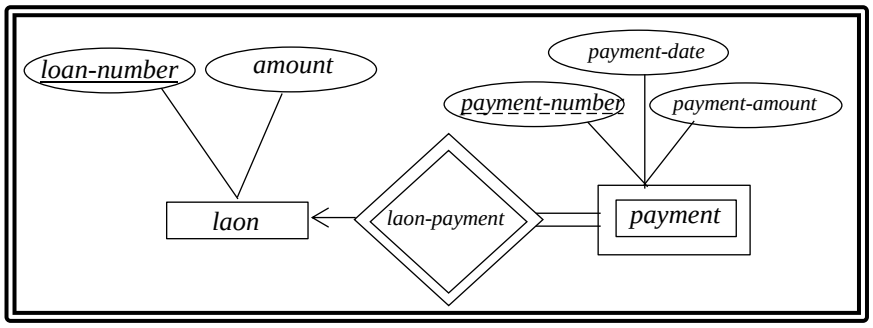
Figure 4.14 the meaning of a relationship



Weak Entity Sets in E-R Diagrams

A weak entity set is indicated by a doubly-outlined box. For example, the weak entity set payment is dependent on the strong entity set loan via the relationship set loan-payment. Figure 4.15 shows this example.

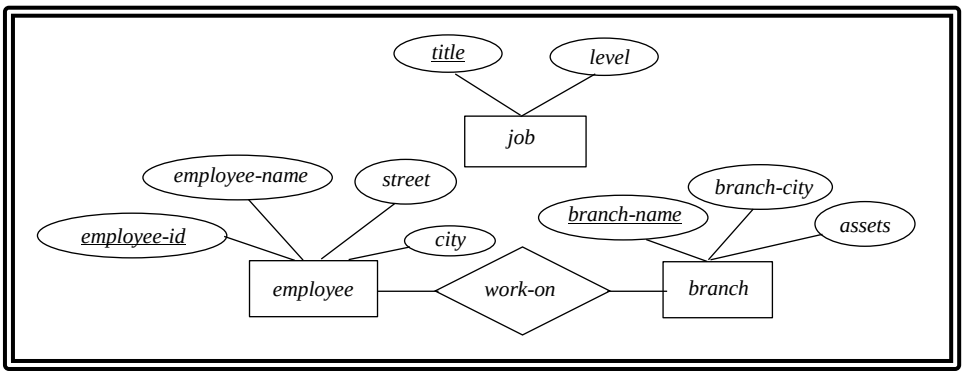
Figure 4.15 the weak entity set



Non-binary Relationships

Non-binary relationships can easily be represented. Figure 4.16 shows an E-R diagram with a ternary relationship.

Figure 4.16 a ternary relationship

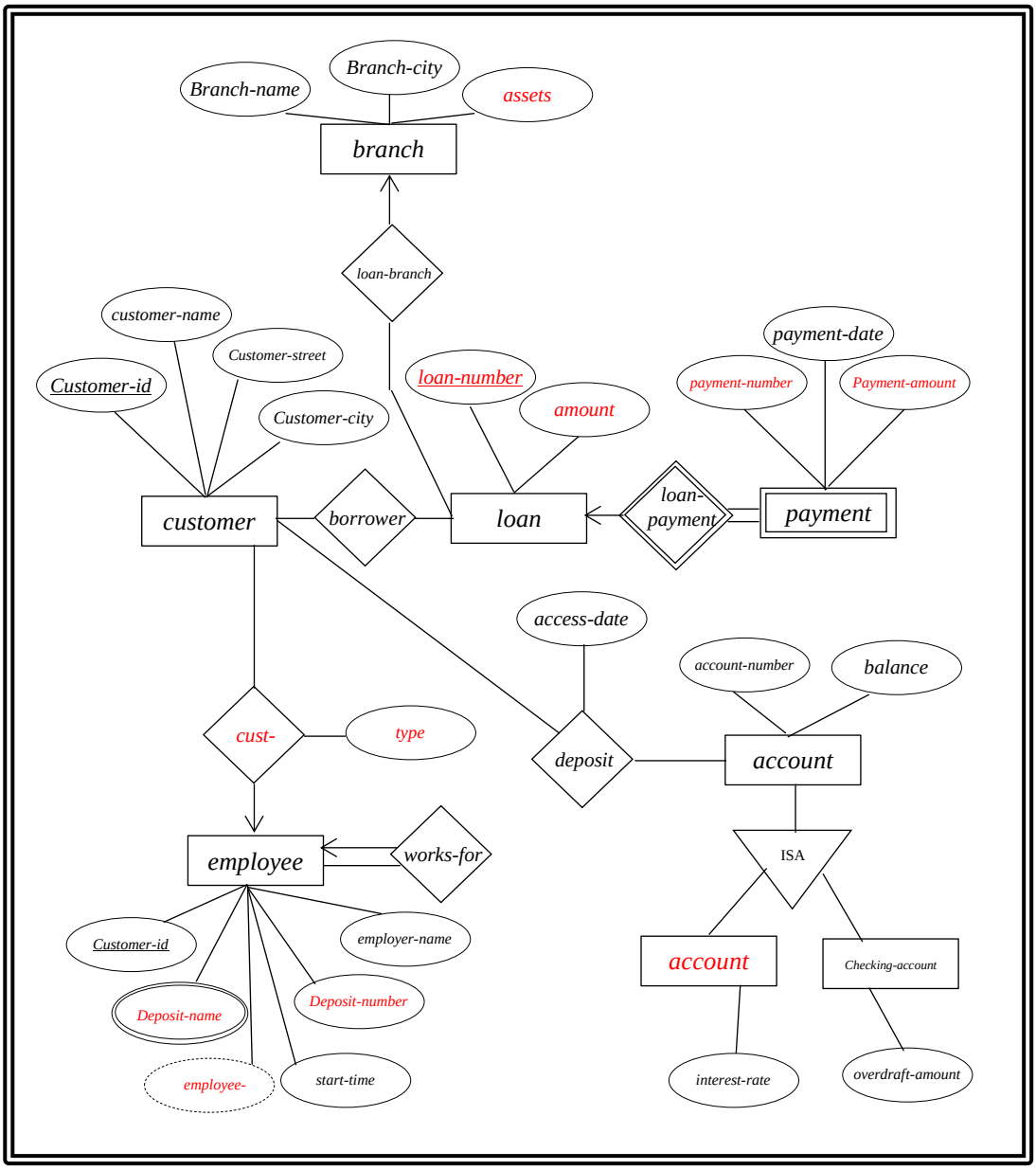


4.6 Reducing E-R Diagrams to Tables

Reduction of ER diagram to Table. The database can be represented using the notations, and these notations can be reduced to a collection of tables.

A database conforming to an E-R diagram can be represented by a collection of tables. For each entity set and relationship set, there is a unique table which is assigned the name of the corresponding set. Each table has a number of columns with unique names. The E-R diagram of Figure 4.17 is used as an example.

Figure 4.17 an E-R diagram for a bank



4.6.1 Strong Entity sets

We use a table with one column for each attribute of the set. Each row in the table corresponds to one entity of the entity set. For the entity set customer, see the table of Figure 4.18.

Figure 4.18 the entity set – customer

customer-id	*customer-name*	*customer-street*	*customer-city*
019-28-3776	Smith	North	Rye
182-73-6091	Turner	Putnam	Stamford
192-83-7465	Johnson	Alma	Palo Alto
244-66-8800	Curry	North	Rye
321-12-3123	Jones	Main	Harrison
335-57-7991	Adams	Spring	Pittsfield
336-66-9999	Lindsay	Park	Pittsfield
677-89-9011	Hayes	Main	Harrison
963-96-3963	Williams	Nassau	Princeton

Weak Entity Sets

For a weak entity set, we add columns to the table corresponding to the primary key of the strong entity set on which the weak set is dependent. For example, the weak entity set payment has four attributes: loan-number, payment-number, payment-date and payment -amount. The primary key of payment is {loan-number, payment -number}. This gives us the table of Figure 4.19.

Figure 4.19 the weak entity set - payment

loan-number	*payment-number*	*payment-date*	*payment-amount*
L-11	53	7 June 2001	125
L-14	69	28 May 2001	500
L-15	22	23 May 2001	300
L-16	58	18 June 2001	135k
L-17	5	10 May 2001	50
L-17	6	7 June 2001	50
L-17	7	17 June 2001	100
L-23	11	17 May 2001	75
L-93	103	3 June 2001	900
L-93	104	13 June 2001	200

4.6.2 Relationship sets

Let R be a relationship set involving entity sets E_1, E_2,.. .., E_m. The table corresponding to the relationship set R has the following attributes:

$$\cup_{i=1}^{m} Primary\_Key(E_i)$$

If the relationship has k descriptive attributes b_1, b_2,…, b_k, we add them too:

$$\cup_{i=1}^{m} Primary_{Key(E_i)} \cup \{b_1, b_2, \ldots b_k$$

An example: The relationship set borrower involves the entity sets customer and loan. Their respective primary keys are customer-id and loan-number. This gives us the table of Figure 4.20.

Figure 4.20 the relationship set – borrower

Customer-id	*loan-number*
019-28-3746	L-11
019-28-3746	L-23
244-66-8800	L-93
321-12-3123	L-17
335-57-7991	L-16
555-55-5555	L-14
677-89-9011	L-15
963-96-3963	L-17

Non-binary Relationship Sets

The ternary relationship of Figure 4.17 gives us the table:

works-on(employ-id, title, branch-name)

As required, we take the primary keys of each entity set. There are no descriptive attributes in this example.

Linking a Weak to a Strong Entity

These relationship sets are many-to-one, and have no descriptive attributes. The primary key of the weak entity set is the primary key of the strong entity set it is existence-dependent on, plus its discriminator.

The table for the relationship set loan-payment in Figure 4.18 would have the same attributes as the table in Figure 4.20, and is thus redundant.

2.6.3 Generalization

We can express the similarities between the entity sets by generalization. This is the process of forming containment relationships between a higher-level entity set and one or more lower-level entity sets.

Consider extending the entity set account by classifying accounts as being either savings-account or checking-account. Each of these is described by the attributes of account plus additional attributes. (Savings has interest-rate and checking has overdraft-amount.) In E-R diagrams, generalization is shown by a triangle ISA relationship set as shown in Figure 4.18.

Generalization hides differences and emphasizes similarities. Distinction made through attribute inheritance: attributes of higher-level entity are inherited by lower level entities. Two methods are available for conversion to a table form, and the following tables are created from the entity sets generalized by the ISA relationship set in Figure 4.18.:

Method 1: Create a table for the high-level entity, plus tables for the lower-level entities containing also their specific attributes.

account(account-number, balance)

savings-account(account-number, interest-rate)

checking-account(account-number, overdraft-amount)

Method 2: Create only tables for the lower-level entities.

savings-account(account-number, balance, interest-rate)

checking-account(account-number, balance, overdraft-amount)

Summary

- *Data modeling is the process of analyzing and defining all the different data your business collects and produces, as well as the relationships between those bits of data. The data modeling the entity-relational mode is intended primarily for the DB design and the entity-relationship model or entity-relationship diagram (ERD) is a data model or diagram for high-level description of conceptual data models.*
- *An entity is a real-world thing or a real-world object which is distinguishable from other objects in the real world. An entity set is a set of entities of the same type. Attribute is a function which maps an entity set into a domain. The domain is the set of permitted values.*
- *A relationship is an association among several entities. A relationship set is a collection of relationships of the same type, and an entity set is a collection of entities of the same type. The terms superkey, candidate key, and primary key apply to entity and relation-ship sets as they do for relation schemas. Identifying the primary key of a relationship set requires some care, since it is composed of attributes from one or more of the related entity sets. Mapping cardinalities express the number of entities to which another entity can be associated via a relationship set. There are 4 mapping cardinality must be one of: one-to-one.one-to-many, many-to-one and many-to-many.*
- *Key is an attribute or a set of attributes that help to uniquely identify a tuple (or row) in a relation (or table) type of Key in DB: Primary Key, Super Key. Candidate Key, Alternate Key, Foreign Key, Composite Key, Unique Key. The terms superkey, candidate key, and primary key apply to entity and relationship sets as they do for relation schemas. Identifying the primary key of a relationship set requires some care, since it is composed of attributes from one or more of the related entity sets.*
- *A database design specified by an E-R diagram can be represented by a collection of relation schemas. For each entity set and for each relationship set in the database, there is a unique relation schema that is assigned the name of the corresponding entity set or relationship set. This forms the basis for deriving a relational database design from an E-R diagram. E-R diagram basic components are: rectangles ellipses, diamonds and lines.*

- *Reduction of ER diagram to Table. The database can be represented using the notations, and these notations can be reduced to a collection of tables.*

Questions

1. What is Entity-Relational Model?
2. What is Entity and Entity Set? Please give example?
3. What is Relationship and Relationship Set?
4. What are 4 Mapping cardinalities?
5. What is Keys?
6. What are types of Keys?
7. What is E-R Diagram?
8. What is Reducing E-R Diagrams to Tables?

Exercises

1. Please look at figure 4.4 and 4.5 mapping cardinalities from textbook and summary of the processing.
2. Please look at figure 4.7 the summary of symbols. Please describe the role of each symbols in the E-R Diagram.
3. Please summary of the role in E-R Diagram.

Chapter 05

SQL

Objective

- To practice DDL: Data Definition Language
- To practice DML: Data Modification

In this chapter, you will learn:

5.1 Introduction

5.2 DDL: Data Definition Language

5.3 DML: Data Modification

5.1 Introduction

① Data Definition Language (DDL)

- Define the structure of the database: CREATE (DATABASE, TABLE, VIEW), DROP, ALTER
- Specify security constraints (authorization): GRANT

② Data Manipulation Language (DML)

- Modify data in the database: INSERT, DELETE, UPDATE
- Query a database: SELECT

• Implemented as a part of the system R project in the early 1970's.

↳ DB2: RDBMS by IBM.

• ANSI published a SQL standard in 1986.

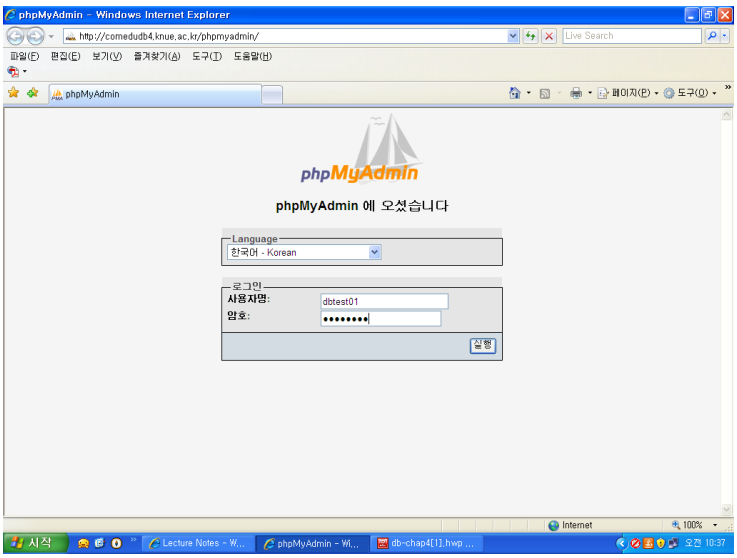
⇒ Then, ANSI and ISO SQL92 standard.

⇒ Every commercial RDBMS supports SQL92.

(DB2, Oracle, Informix, Sybase, MS SQL-Server, MySQL etc.)

• MySQL: phpMyAdmin

- MySQL User ID/Password



5.2 DDL: Data Definition Language

The most common command types in DDL are CREATE, ALTER and DROP. All three types have a predefined syntax that must be followed for the command to run and changes to take effect.

• Used by DBA.

• Creating a DB

CREATE DATABASE D (…)

• Creating a relation into the DB

CREATE TABLE R (A1 D1, A2 D2, … , An Dn)

WHERE R: the name of the relation

Ai: the attribute

Di: the data type of the domain Ai

Customer

name	city	address
char(20)	char(20)	char(50)

```
CREATE TABLE Customer
        (name char(20) NOT NULL,
    city char(20),
        address  char(50),
        PRIMARY KEY (name))
```

Branch

branch	asset	city
char(15)	int	char(30)

```
CREATE TABLE Branch
    (branch char(15) NOT NULL,
     asset  integer,
     city   char(30),
     PRIMARY KEY (branch))
```

Deposit

branch	account	name	balance
char(15)	char(10)	char(20)	int

```
CREATE TABLE  Deposit
       (branch   char(15),
        account char(10) NOT NULL,
        name   char(20),
        balance   integer,
                PRIMARY KEY (account),
                CHECK (balance>=100),
               FOREIGN KEY (branch) REFERENCES Branch,
               FOREIGN KEY (name) REFERENCES Customer)

CREATE TABLE Laon
    (branch char(15),
    account char(10) NOT NULL,
    name char(20),
    balance integer,
                PRIMARY KEY (account),
                CHECK (balance>=100),
                FOREIGN KEY (branch) REFERENCES Branch,
                FOREIGN KEY (name) REFERENCES Customer);
```

Entity integrity?

Domain integrity?

Referential integrity?

• Create an index on the attribute

```
CREATE INDEX A-ix ON R(A)
CREATE INDEX AB-ix ON R(A,B)
```

• Remove a relation from the DB

```
DROP TABLE R        /* Remove R from the DB. */
```

• Change a relation scheme.

```
ALTER TABLE R     ADD   A D
                        MODIFY
                        DROP
```

e.g., ALTER TABLE R ADD A char(15)

ALTER TABLE R MODIFY A integer

ALTER TABLE R DROP Primary Key

→ ALTER TABLE R ADD Primary Key R(A)

e.g., ALTER TABLE dept DROP PRIMARY KEY CASCADE;

⇒ The CASCADE option drops any 'foreign keys' that reference the primary key

• Creating a View (NOT a relation)

- Security considerations may require that certain data be hidden from user.

⇒ It is possible to support a large number of views on top of any given set of actual relations.

⇒ A view must be 'recomputed' for each query that refers to it.

e.g., CREATE VIEW DepositCustomer AS

(SELECT account, branch, name FROM Deposit)

e.g., CREATE VIEW AllCustomer AS

(SELECT branch, name FROM Deposit)

UNION

(SELECT branch, name FROM Loan)

5.3 Data Manipulaltion Language (DML)

DML is an abbreviation for Data Manipulation Language. Data Manipulation Language or DML represents a collection of programming languages explicitly used to make changes

in the database, such as: CRUD operations to create, read, update, and delete data. Using the INSERT, SELECT, UPDATE and Delete commands

• Used by programmers.

• Insertion

- A newly created relation is empty initially.

⇒ Load data into the relation.

Deposit

branch	account	name	balance
kangnam	123	YS	1000

Loan

branch	account	name	balance

INSERT INTO Deposit
VALUES (‘kangnam‘, 123, ‘YS‘, 1000)

INSERT INTO (account, name, balance, branch)
VALUES (123, ‘YS‘, 1000, ‘kangnam‘)

INSERT INTO Deposit
(SELECT branch, account, name, 200
FROM Loan
WHERE branch=‘kangnam’)

INSERT INTO Deposit
VALUES (‘kangnam‘, 123, NULL, 1000) /* Not yet determined (≠0) */

-An insertion is permitted through a view only if the view in query is defined in terms of one relation of the actual DB.

e.g., INSERT INTO DepositCustomer VALUES (‘kangnam‘, 111, ‘Rho‘)

⇒ (‘kangnam‘, 111, ‘Rho‘, NULL) is inserted into Deposit.

What if All Customer?

- All aggregate operation(AVG, MIN, MAX, SUM, COUNT) ignore tuples with NULL values on the argument attributes.

e.g., SELECT SUM(balance)
FROM Deposit

SELECT COUNT(name) AS deposit Customer
FROM Deposit

• Updating: Change a value in a tuple

e.g., UPDATE Deposit SET balance=balance*1.05 /* interest rate= 5%*/

e.g., UPDATE Deposit SET balance=balance*1.06
WHERE balance>10000;
UPDATE Deposit SET balance=balance*1.05
WHERE balance<=10000

• Deletion

e.g, DELETE FROM Loan /* delete all tuples */

DELETE FROM Loan WHERE name=‘YS‘

DELETE FROM Loan WHERE account BETWEEN 100 AND 200

DELETE FROM Loan WHERE branch IN (SELECT branch
FROM Branch
WHERE city=‘seoul‘)

DELETE FROM Loan WHERE balance < (SELECT AVG(balance)
FROM Deposit)

⇒ ① Calculate average.
② Mark tuples to be deleted.

③ Once done marking, delete all marked tuples.

[Example: Bank DB]

Branch

branch	asset	city
kwangan	100	pusan
haundae	200	pusan
dongback	400	pusan
sinchon	300	seoul
kangnam	50	seoul
seocho	500	seoul

Customer

name	city	address
YS	pusan	kwangan
Rho	pusan	haundae
Chun	pusan	dongback
DJ	pusan	kwangan

Deposit

branch	account	name	balance
kwangan	111	YS	100
kwangan	112	Rho	200
kwangan	113	Chun	400

Loan

branch	account	name	balance
kwangan	111	YS	10000
haundae	112	Rho	20000
dongback	113	Chun	40000
kwangan	114	DJ	50000

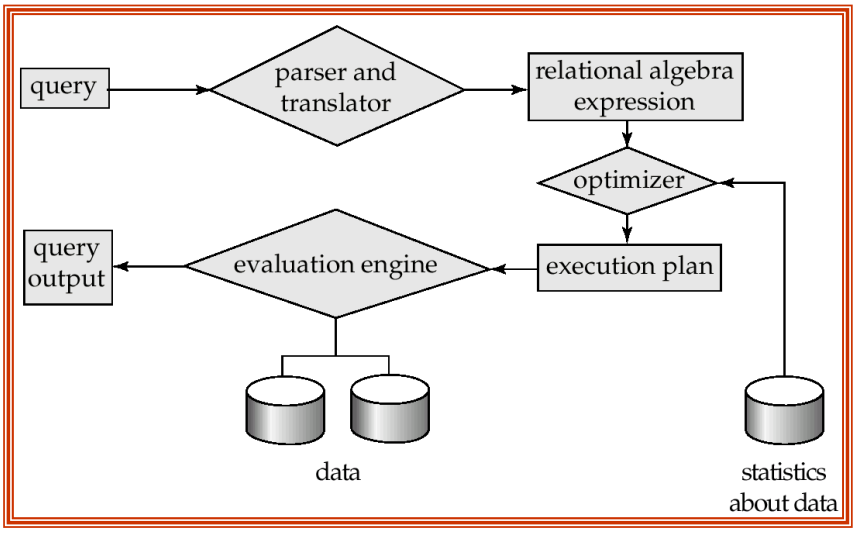
```
desc Branch;
desc Customer;
desc Deposit;
desc Loan;

select * from Branch;
select * from Customer;
select * from Deposit;
select * from Loan;
```

• Query Processing

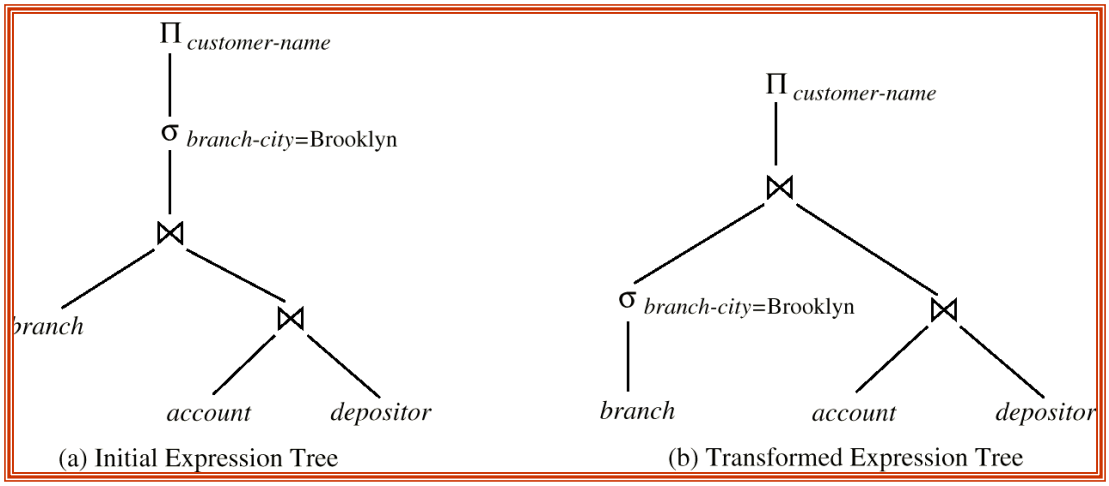
1. Translation of a high-level SQL query
   ⇒ Internal ‘relational algebra’ expression

2. Query optimization: The most efficient execution plan
   ⇒ Performance in terms of the 'no. of disk accesses' required
   ⇒ The difference between a 'bad' strategy and a 'good' strategy

1. Execution of a query: Query output



• Query-execution plan
- Parse-tree representation
- Annotated instruction



• Retrieving

```
SELECT A1, A2, ... , An
        FROM R1, R2, ... , Rm
                WHERE P
        where  Ai: attribute
                Ri: relation
                P : predicate
```

```
SELECT *                /* project all attributes of all relations */
    FROM  R1, R2, ... , Rm   /* in the FROM clause.              */
    WHERE P
```

Duplicate tuples

```
SELECT DISTINCT branch   /* Duplicates are removed */
        FROM Deposit

    SELECT branch       /* Duplicates are NOT removed */
        FROM Deposit
```

Set operations

A list of customers who have a deposit account, a loan account, or both accounts at the ‘kwangan’ branch?

```
(SELECT name
        FROM Deposit
        WHERE branch=‘kwangan‘)
         UNION
      (SELECT name
        FROM Loan
        WHERE branch=‘sinchon‘)
```

A list of customers who have both accounts in the ‘kwangan’ branch?

```
(SELECT name
  FROM Deposit
  WHERE branch=‘kwangan‘)
   INTERSECT
(SELECT name
  FROM Loan
  WHERE branch=‘sinchon‘)
```

A list of customers who only have a loan account at the ‘kwangan’ branch?

⇒ MINUS (NOT standard, Supported in System R)

⇒ EXCEPT

A list and city of customers who have a loan account?

Join

```
SELECT Customer.name, city
        FROM Customer, Loan
        WHERE Customer.name = Loan.name  // join attr.

⇔ SELECT Customer.name, city
        FROM Customer natural join Loan              // natural join

⇔ SELECT Customer.name, city
        FROM Customer join Loan using (name)         // join ~ using (join attr.)
```

The list and city of customers who have a loan account at the ‘kwangan’ branch?

```
SELECT Customer.name, city
        FROM Customer, Loan
        WHERE Customer.name = Loan.name and branch=’kwangan’
```

A list of customers who have a loan account at the ‘kwangan’ branch, city and balance of deposit?

```
SELECT Customer.name, city
        FROM Customer, Loan
        WHERE Customer.name = Loan.name and Customer.name = Deposit.name and
        branch=’kwangan’
```

A list of customers who have both accounts in the ’kwangan’ branch?

⇒ INTERSECT

```
SELECT Loan.name
        FROM Loan JOIN Deposit USING (name)      /* Natural Join */
        WHERE Loan.branch = 'kwangan'
```

(c.f.) SELECT Deposit.name
FROM Loan JOIN Deposit USING (name)
WHERE Deposit.branch = 'kwangan'

(c.f.) SELECT Deposit.name
FROM FROM Loan JOIN Deposit USING (name, branch)
WHERE Deposit.branch = 'kwangan'

Arithmetic Expression

SELECT account
FROM Deposit
WHERE balance BETWEEN 200 AND 300
(⇔ WHERE balance >= 200 AND balance <= 300)

SELECT account
FROM Deposit
WHERE balance NOT BETWEEN 200 AND 300

String match

Any substring:	% Oracle	* Informix	% MySQL
Any character:	-	?	_ (underbar)

SELECT name
FROM customer
WHERE address LIKE ‘%kwang%‘

SELECT name
FROM customer
WHERE address NOT LIKE ‘%kwang%‘

escape character: ‘percent: \%‘

Set Membership (∈)

⇒ INTERSECT

```
SELECT DISTINCT name
        FROM Loan
        WHERE branch='kwangan' and
            name IN (SELECT name
                    FROM Deposit
                    WHERE branch='kwangan')
```

⇔

```
SELECT DISTINCT name
        FROM Loan
        WHERE branch='kwangan' AND
            (branch, name) IN (SELECT branch, name
                                FROM Deposit)
```

Set comparison

Which branch has greater assets than the branches in 'Pusan'?

Which branch has greater assets than all branches in 'Pusan'?

```
SELECT branch
        FROM Branch
        WHERE  city != 'Pusan' and      /*  <> */
          asset 〉SOME (SELECT asset
                        FROM branch
                        WHERE city='Pusan')
```

```
SELECT branch
        FROM Branch
        WHERE  city != 'Pusan' and      /*  <> */
          asset 〉ALL (SELECT asset
                        FROM branch
```

```
                    WHERE city='Pusan')
```

Testing for empty relations.

EXISTS → True : if the subquery is non empty.

→ False : 〃 empty.

A list of customers who have both deposit and loan accounts in the 'kwangan' branch?

① Set operation: INTERSECT

② Join

③ Set membership: IN

④ EXISTS

```
SELECT name
        FROM Customer
        WHERE EXISTS (SELECT *  FROM Deposit
                        WHERE Deposit.name=Customer.name and
                            branch='kwangan')
         AND EXISTS (SELECT *  FROM Loan
                        WHERE Loan.name=Customer.name and
                            branch='kwangan')
```

Tuple variables: Rename (ρ)

Tuple variables are defined in the FROM clause.

⇒ Tuple variables are most useful for comparing two tuples in the same relation.

The name of the customer who has a savings account at the same branch as 'YS'?

```
SELECT DISTINCT T.name
        FROM Deposit S, Deposit T
        WHERE S.name ='YS'  and
            S.branch = T.branch AND
            T.name != 'YS'
```

```
⇔ SELECT DISTINCT name
        FROM Deposit
        WHERE S.name =‘YS‘  and
            branch IN ( SELECT branch FROM Deposit
                WHERE name = ‘YS‘)
```

Ordering the tuples

```
SELECT name
        FROM   Deposit
        WHERE   branch=‘kwangan‘
        ORDER BY name  ASC   /* DESC  */
```

Aggregate functions

AVG, MIN, MAX, SUM, COUNT

```
SELECT COUNT(name) FROM Deposit

SELECT SUM(balance) FROM Deposit

SELECT branch, AVG(balance) FROM Loan
      GROUP BY branch

CREATE VIEW View30000 AS
(SELECT branch, AVG(balance) AS ‘avg loan amount’,
      SUM(balance) AS 'total loan amount'
            FROM Loan
            GROUP BY branch
            HAVING AVG(balance) >= 30000)
```

Outer join

⇒ Include (left/right/full) tuples not included in natural join

```
NATURAL LEFT OUTER JOIN
```

NATURAL RIGHT OUTER JOIN
NATURAL FULL OUTER JOIN

Referential integrity

```
CREATE TABLE  Employee
      (name   char(10),
       branch   char(10),
       FOREIGN KEY (branch) REFERENCES Branch);

INSERT INTO Employee VALUES ('An‘, ’kwangan‘);
INSERT INTO Employee VALUES ('Yu‘, ’onchun‘);
```

Domain integrity

```
/* CHECK (balance>=100) */

INSERT INTO Deposit
          VALUES (‘kwangan‘, 123, ‘DJ‘, 10)
```

Summary

- *The most common command types in DDL are CREATE, ALTER and DROP. All three types have a predefined syntax that must be followed for the command to run and changes to take effect.*
- *DML is an abbreviation for Data Manipulation Language. Data Manipulation Language or DML represents a collection of programming languages explicitly used to make changes in the database, such as: CRUD operations to create, read, update, and delete data. Using the INSERT, SELECT, UPDATE and Delete commands.*

Questions

1. What is DDL: Data Definition Language command in SQL?
2. What is DML : Data Manipulation Language command in SQL
3. What are the difference between DDL: Data Definition Language and DML: Data Manipulation Language in SQL.

Exercises

1. Please write the correct SQL statement to create a new database called rttcDB.
2. Please write the correct SQL statement to create table grade follow table below:

Field	*Type*	*Null*	*Key*	*Default*
ID	*Integer(3)*	*Yes*	*Primary Key*	*Null*
khmer	*Integer(3)*	*Yes*		*Null*
math	*Integer(3)*	*Yes*		*Null*
science	*Var(3)*	*Yes*		*Null*

- Please write the correct SQL statement to insert 3 students into table grade follow table below:

ID	*khmer*	*math*	*science*	*comment*
111	*90*	*95*	*85*	*outstanding*
112	*100*	*95*	*75*	*kind warm*
113	*70*	*90*	*80*	*need to study hard*

3. Please write the correct SQL statement to create table student follow table below:

Field	*Type*	*Null*	*Key*	*Default*
ID	*Integer(3)*	*Yes*	*Primary Key*	*Null*
name	*var(50)*	*Yes*		*Null*
address	*var(50)*	*Yes*		*Null*

- Please write the correct SQL statement to insert 3 students into table student follow table below:

ID	*name*	*address*
111	*Chan dara*	*Null*
112	*Uk bora*	*Null*
113	*Sorn piseth*	*Null*

4. Please write the correct SQL statement update table grade set math=60 with Id 112
5. Please write the correct SQL statement select from grade by ordering from large to small through math.
6. Please write the correct SQL statement select from grade by aggregate Functions through khmer.
7. Please write the correct SQL statement select student name with the highest khmer score.
8. Please write the correct SQL statement select the name of the lowest student and show all the scores that get comment study hard.
9. Please write the correct SQL statement Joined Relations (Inner join, Left outer join, Natural inner join, Natural full outer join) and show result?

Chapter 06

PHP

Objective

- To Understand World Wide Web
- To practice Web Server(httpd)
- To practice PHP (Professional Hypertext Preprocessor)
- To practice My SQL

In this chapter, you will learn:

6.1 WWW (World Wide Web)

6.2 Installing Web Server (httpd)

6.3 Installing PHP (Professional Hypertext Preprocessor)

6.4 Install My SQL

6.1 WWW (World Wide Web)

• Introduction

- Wide-area information service and software that allows you to search all kinds of information distributed on the Internet in a unified way
- Proposed by Tim Berners-Lee of the European Council for Nuclear Research (CERN) in 1989.
- Stored and managed as a unit called Home Page in the form of Hypertext within the Web Server
- Connected to Hypertext around the world geographically distributed on the Internet by a function called Link.

• HTTP (HyperText Transfer Protocol)

- Standard protocol that defines how a web client and web server communicate and how data information is transferred from the web server to the web client.
- Obtain data information of web server by entering URL (Uniform Resource Locator) starting with http://
- TCP/IP is used for the correct transmission of data in the underlying

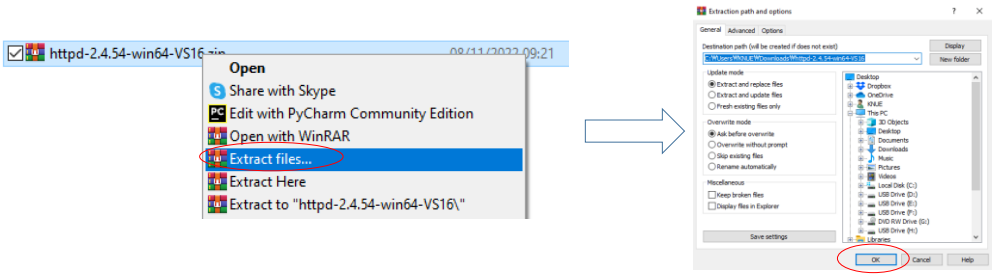
6.2 Installing Web Server (httpd)

Apache is the most commonly used Web server on Linux systems. Web servers are used to serve Web pages requested by client computers. Clients typically request and view Web pages using Web browser applications such as Firefox, Opera, Chromium, or Internet Explorer.

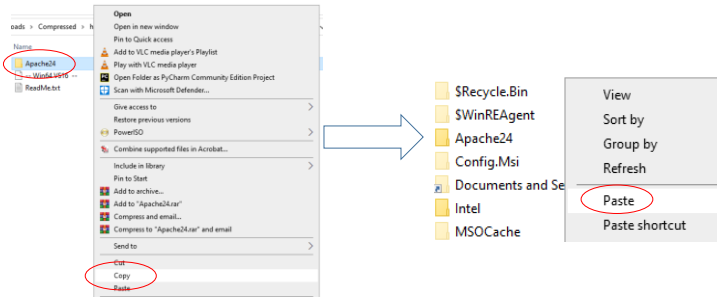
• Installing Apache on Windows 10 64bit Environment

- Go to '**https://www.apachelounge.com/download/**' and download it.

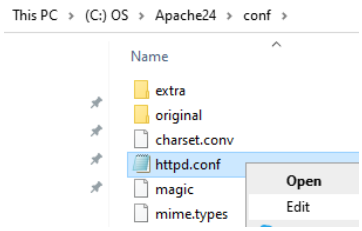
- Unzip the downloaded file.



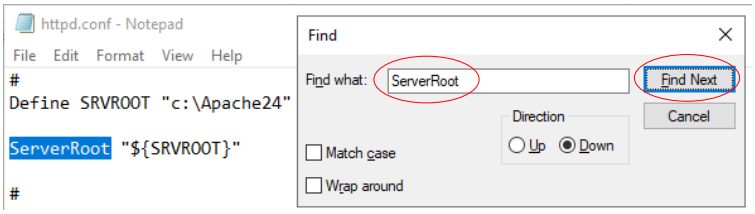
- Move the Apache24 folder to the 'C:\'. After moving, the final path will be '**C:\Apache24'** path.



- Open the 'httpd.conf' file with notepad. Basically, find the part below and change the settings to suit your installation PC. (Specifies the Server Root path. Since it was specified as C:\Apache24 above, modify it accordingly.)



ServerRoot “c:\Apache24”



Note: Press Ctrl+F, and then type your search words

- Set the port of the web server. The default is 80. If you do not use a different port number, leave it as is.

Listen 80

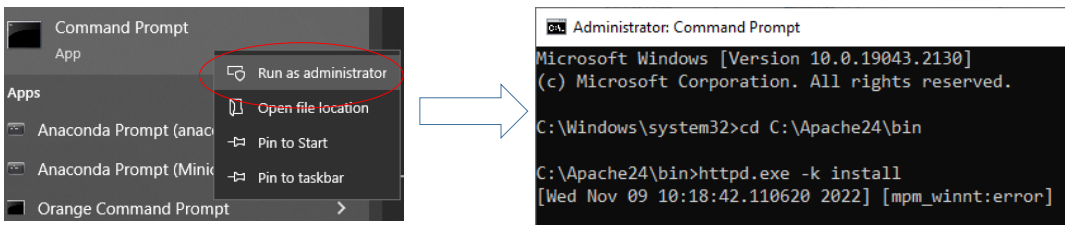
- This is the path where the files of the website displayed when accessing the web server with a web browser are saved. When connecting to http://localhost:80 (or http://localhost), the index.html page in DocumentRoot is found and displayed.

DocumentRoot “c:\Apache24/htdocs”

- Run the command prompt window in administrator mode. (If you want to uninstall the installed Apache server, type **httpd.exe -k uninstall** command.)

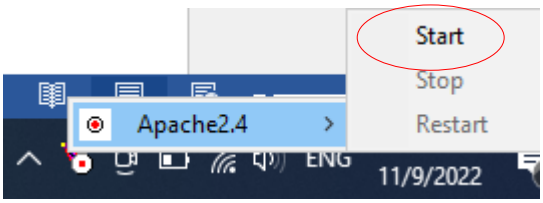
C:\Apache25/bin

Httpd.ext –k install



- Execute the file below in Window File Manager:

C:\Apache24\bin\ApacheMonitor.exe



- Run ‘Apache Monitor’ by right-clicking the Apache icon on the taskbar at the bottom right of the Window.

- Start and stop the server using the start and stop buttons in Apache Monitor.
- Launch a web browser and connect to ‘http://localhost’.
- The will produce the following Result:

6.3 Installing PHP (Professional Hypertext Preprocessor)

PHP (Hypertext Preprocessor) is known as a general-purpose scripting language that can be used to develop dynamic and interactive websites. It was among the first server-side languages that could be embedded into HTML, making it easier to add functionality to web pages without needing to call external files for data.

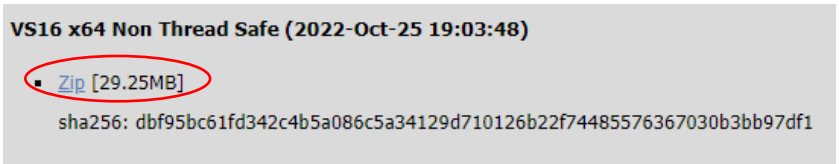
• Introduction

- Server-side script: A technology that directly processes the files coded in the web server by the application server operating on the server side and sends the results to the browser
- You can configure the fastest website running on Windows, Unix, and Linux operating systems.
- Since it is a language developed with an open source model, it is ported so that it can be operated in various operating systems and web server environments.
- A very accessible web development language for small developers.

• Installing PHP and connecting to httpd

- Please download the appropriate version of PHP from the site below. You must select ‘Thread Safe’ rather than Non Thread Safe for it to work properly! (the VS16 version of 64bit Thread Safe)

windows.php.net/download/



- Change the name to php8 (php version name) and save it in the same folder as Apache, and change 'php.ini-development' in the folder to '**php.ini'**.
- After that, open php.ini with notepad.
- Find extension_dir and change **extension_dir = "C:\php8\ext**",

extension=curl
extension=mysqli
extension=gettext
extension=mbstring
extension=openssl
extension=pdo_sqlite

- remove the semicolon ';' in front of it to activate the module.
- Additionally, to set the time zone, search for [Date] and change **'date.timezone'** as follows.

[Date]
; Defines the default timezone used by the date functions
; http://php.net/date.timezone
date.timezone = Asia/Seoul

- Change the error reporting settings as follows.

error_reporting = E_ALL & ~E_NOTICE & ~E_DEPRECATED & ~E_USER_DEPRECATED

- After that, open '**httpd.conf'** in conf in Apache24 folder with notepad and paste the following part in the bottom line.

PHPIniDir "C:/php8"
LoadModule php_module "C:/php8/php8apache2_4.dll"
AddHandler application/x-httpd-php .php
AddType application/x-httpd-php .php .html

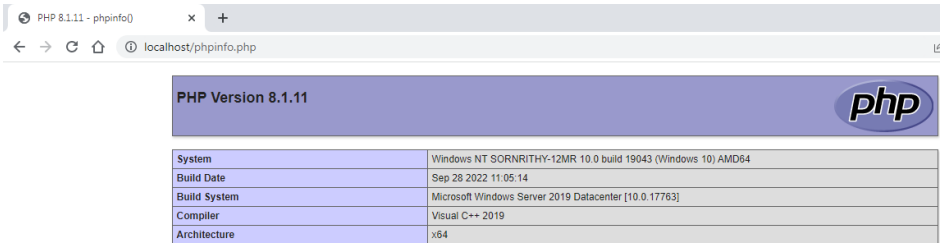
- Search for '**IfModule dir_module'** and replace it as follows.

<IfModule dir_module>
DirectoryIndex index.php index.html
</IfModule>

- Finally, create '**phpinfo.php'** as below in 'Apache24\htodcs'.

```
<?php
    phpinfo();
?>
```

- After that, type **'localhost/phpinfo.php'** in the address bar and the screen of php information will appear!
- The will produce the following Result:

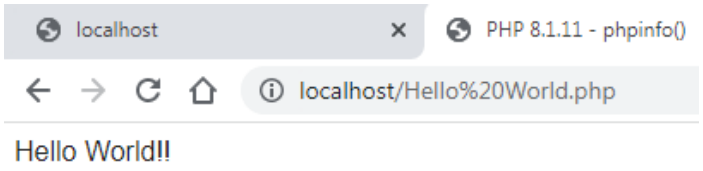


Basic Programming

• php #1-1: Hello World

```
<?php
    Echo "Hello World!! \n";
        phpinfo();
?>
```

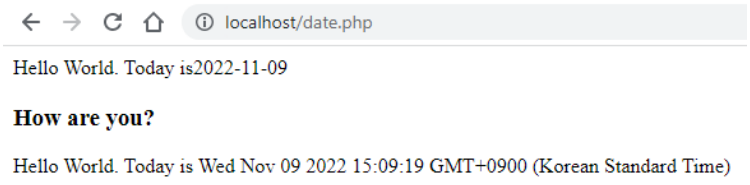
- The will produce the following Result:



• php #2-1: php, html, Javascript

```
<?php
Echo "Hello World. Today is".date("Y-m-d", time());
?>
<h3>How are you?</h3>
<script>
        document.write("Hello World. Today is "+Date());
</script>
```

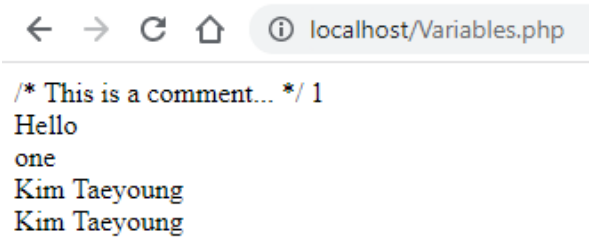
- The will produce the following Result:



• php #3-1: variables

```
/* This is a comment... */
$mycounter=1;
$mystring="Hello";
$myarray=array("one","two","three");
echo $mycounter."<br>";
echo $mystring;
echo "<br>";
echo $myarray[0];
echo "<br>";

$username="Kim Taeyoung";
echo $username;
echo "<br>";
$current_user=$username;
echo $current_user;
?>
```

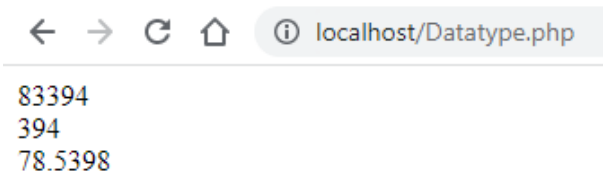
- The will produce the following Result:



• php #3-2: types

```
<?php
/* automatic transformation... */

$number=123*678;
echo $number."<br>"
echo substr($number,3,2)."<br>";

$pi="3.141592";
$radius=5;
$area=$pi*($radius*$radius);
echo $area;
?>
```

- The will produce the following Result:

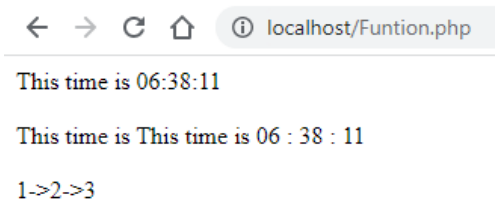


• php #3-3: function

```
<?php
/* function... */

$temp="The time is ";
echo $temp.longdate1(time());

function longdate1($timestamp){
        return date("H:i:m", $timestamp);
}

echo "<br><br>";
```

```
echo longdate2($temp,time());

function longdate2($text,$timestamp){
        return $text.date("H : i : m", $timestamp);
}

echo "<br><br>";
echo test()."->".test()."->"test();
function test(){
        static $count=0;
        $count++;
        return $count;
}
?>
```

- The will produce the following Result:

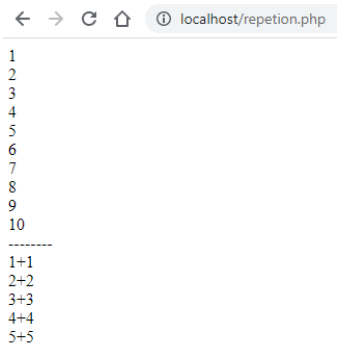


• php #4-1: repetition

```
<?php
/* This is a comment... */

$count=0;
while(++$count<=10)
        echo $count."<br>";

echo "--------<br>";
for($i=1, $j=1; $i+$j<=10; $i++, $j++)
        echo $i."+".$j."<br>";
```

```
$fp=fopen("4-1.txt","wb");
for($j=0; $j<10; ++$j){
        $written=fwrite($fp, "data".$j."\n");
        if($written==FALSE) break;
}
fclose($fp);
?>
```

- The will produce the following Result:



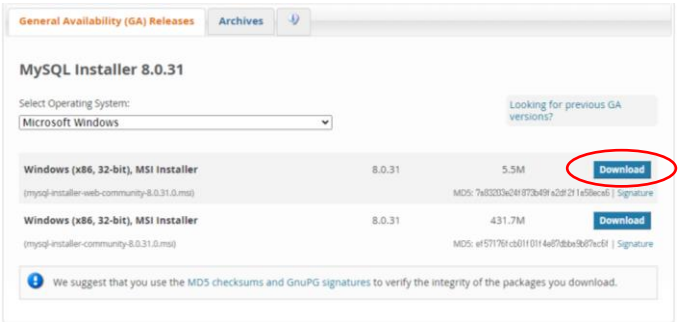
6.4 Installing MySQL

MySQL is a tool used to manage databases and servers, so while it's not a database, it's widely used in relation to managing and organising data in databases.

• Installing MySQL

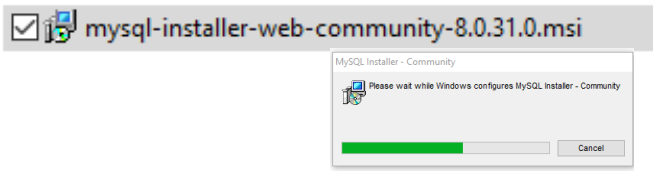
https://dev.mysql.com/downloads/installer/

First of all, we need to download the installer.



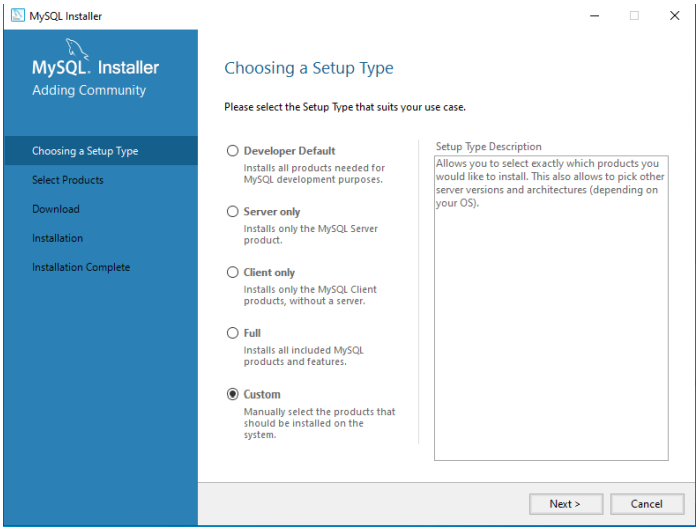
When you try to start the download, the website will ask you to log in or create an account, but you don't have to do so. Note the ***No thanks, just start my download*** button.

When you open the installer, it will first configure the installation and then ask for the user's permission to proceed:



When this is over, we'll finally see the installer interface. As you can see in the image below, the process consists of four steps:

1. Choosing a setup type
2. Downloading the files
3. Installing the software
4. Finishing the installation



There are five types of setups available in this first step and you can check the side box to see what each of them will install. However, we strongly recommend, especially if you're just getting started with SQL, to select the default option.

For select the default option: The **default option** is large. It requires a large computer to use to install Mysql to run properly and avoid crashes during use.

The most important features, among others, this setup will install are:

- MySQL Server: the database server itself
- MySQL Workbench: an application to manage the server
- MySQL for Visual Studio: this feature enables the users to use MySQL from Visual Studio

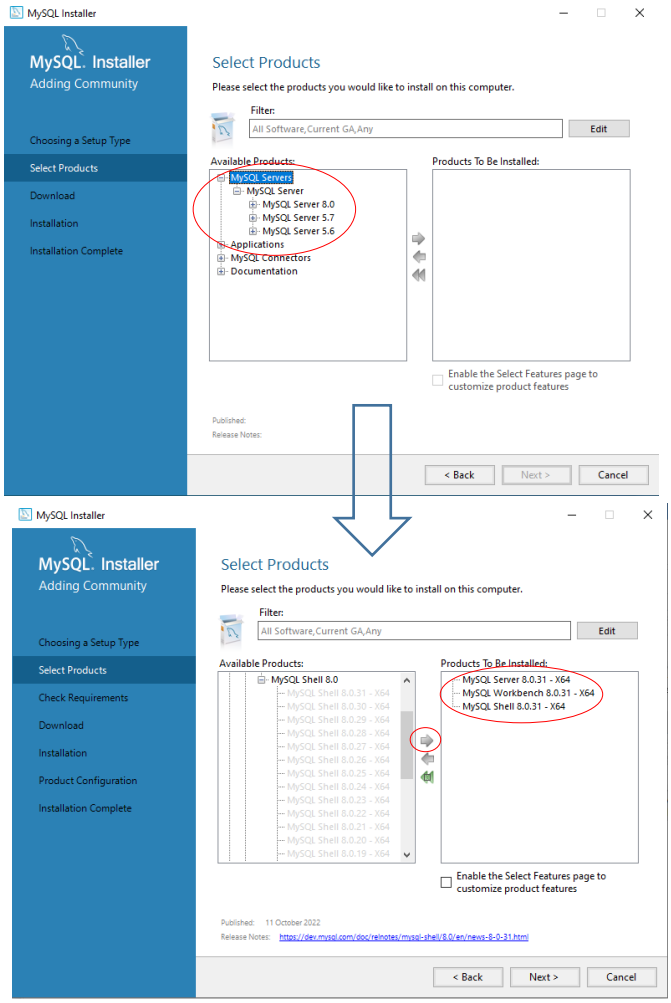
- The documentation and tutorials

It's also ok to choose the full setup as this will install all MySQL resources available.

But if your computer is medium or under we should **choose the Custom** and select the products you would like to install on your computer.

After you choose the setup option, click on **Next**.

If we select Custom please do the following:



1. Choose the setup: MySQL Server 8.0.31 – X64

- Click on the + MySQL servers
- Click on the + MySQL Server
- Click on the + MySQL Server 8.0
- Select MySQL Server 8.0.31 – X64 and press the right arrow

2. Choose the setup: MySQL Workbench 8.0.31 – X64

- Click on the + Application
- Click on the + MySQL Workbench

- Click one the + MySQL Workbench 8.0
- Select MySQL Workbench 8.0.31 – X64 and press the right arrow

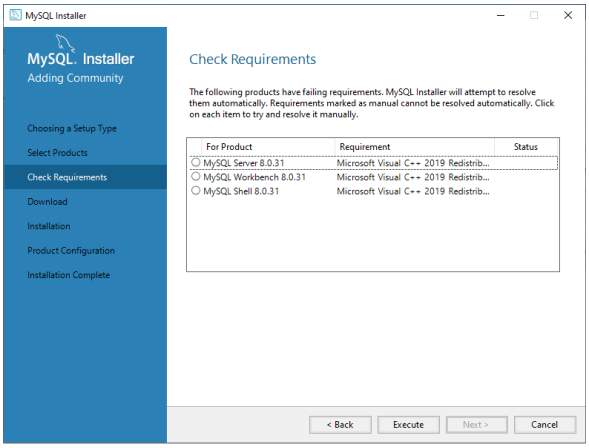
3. Choose the setup: MySQL Shell 8.0.31 – X64

- Click on the + Application
- Click on the + MySQL Shell
- Click on the + MySQL Shell 8.0
- Select MySQL Shell 8.0.31 – X64 and press the right arrow

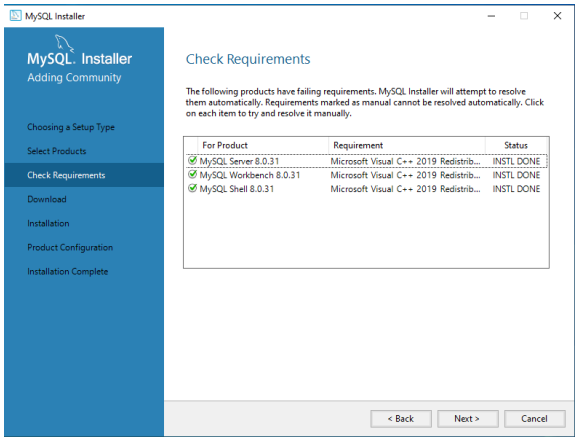
Click on **Next**

Requirements

At this point, there's a chance you'll be asked to install some required software, the most common being the Visual Code. The installer can automatically solve some requirement issues, however, this is not the case here:



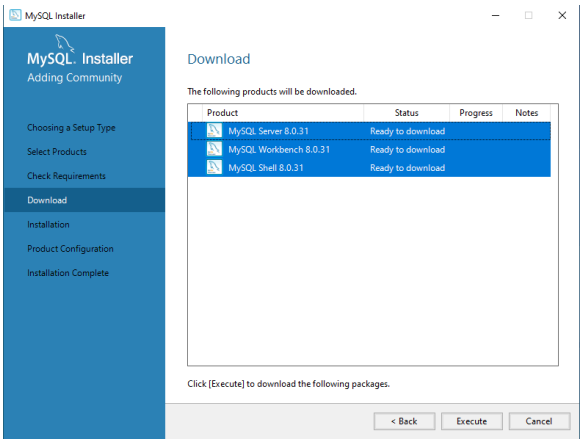
- Click on **Execute**
- √ I agree to the licences terms and conditions
- Click on **Install**
- Setup Successful (Click on **Close**)



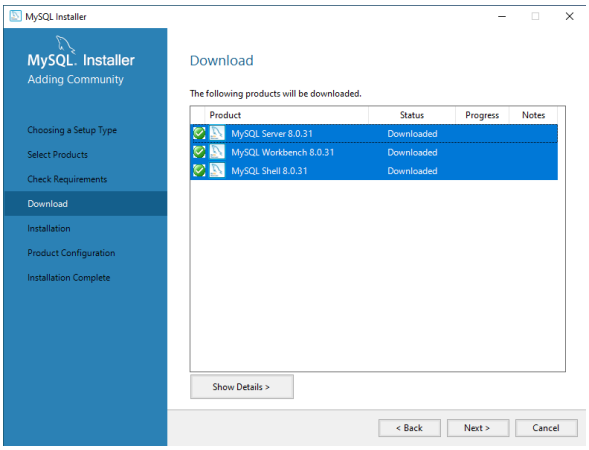
Click on **Next**

Download & Install

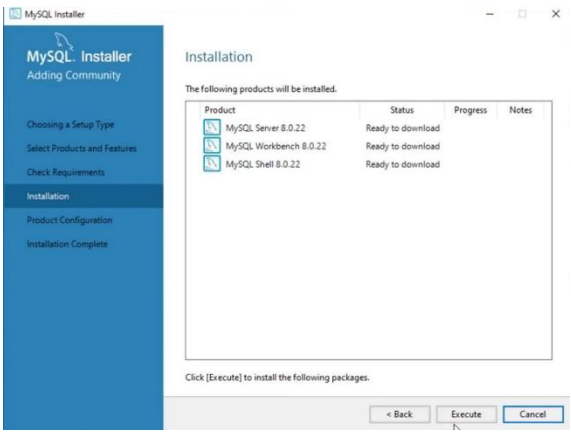
You have now reached the download section. The section name is self-explanatory: you'll download all the components in the setup option you selected.



Click on **Execute** and the download will start. This might take a few minutes to be concluded. When it's done, you should see tick marks on every item. Then you can proceed.



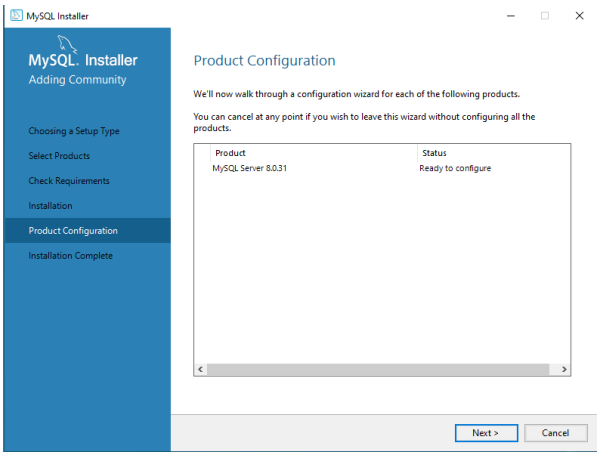
The next screen you'll see is almost the same as the last one, but now it will install all the components you've just downloaded. This step will take significantly longer than the previous one. When it's over, you'll see all the ticks marks again:



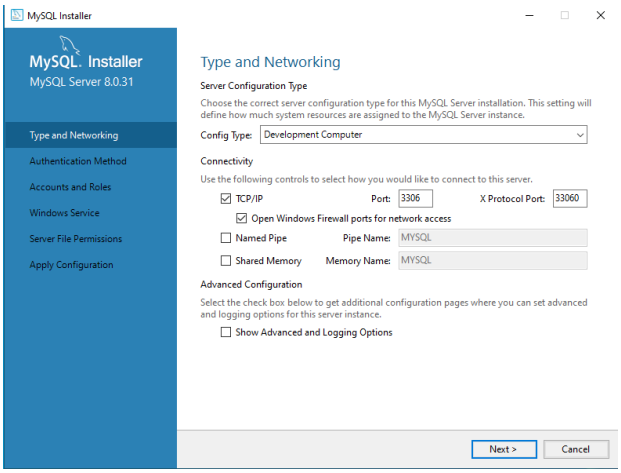
Click on **Execute**

Configuration

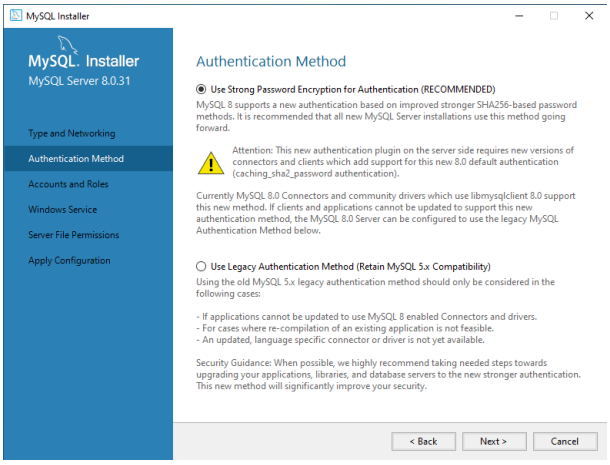
The next step is to configure the server. You'll see the following screen. Hit **Next**.



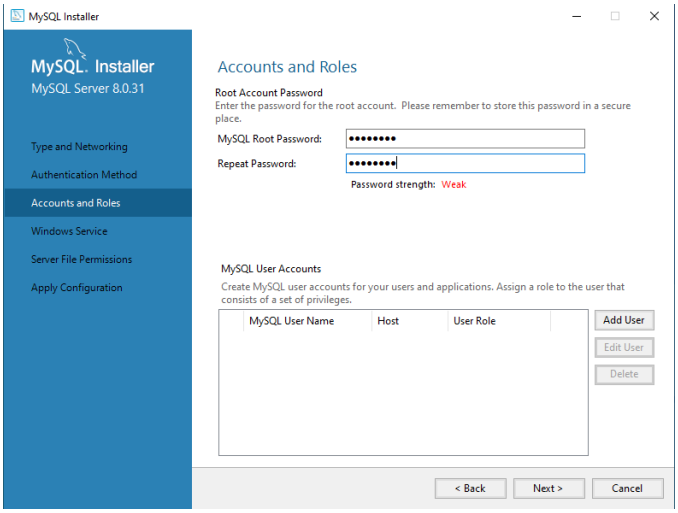
First, the installer will ask you to configure the network:



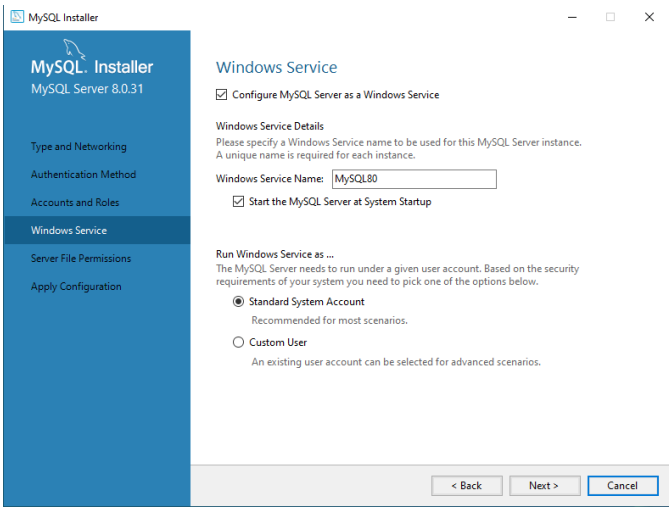
It's important to keep *Development Computer* in the *Config Type* field as you're probably installing this on your personal computer and not on a dedicated machine. You can choose the port, but the default will work just fine. Click **Next**. For the authentication method, let's stick with the recommended option and click **Next**:



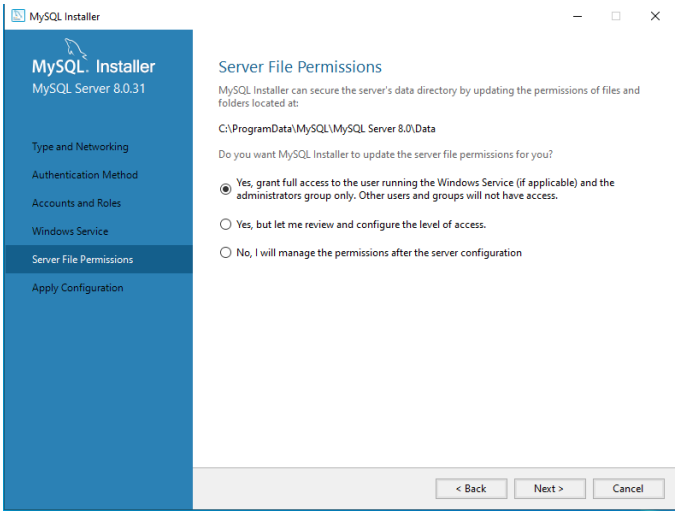
Now it’s time to create the root account. You’ll be asked to set a password. Remember to use a strong one.
On this same screen, you can create other users and set their passwords and permissions. You just have to click on **Add User** and fill in the blanks. Then, click **Next**.



Now you can choose the Windows service details, such as the service name, account type, and if you want to start MySQL when you turn on your computer. Again, the default options will work in most cases:

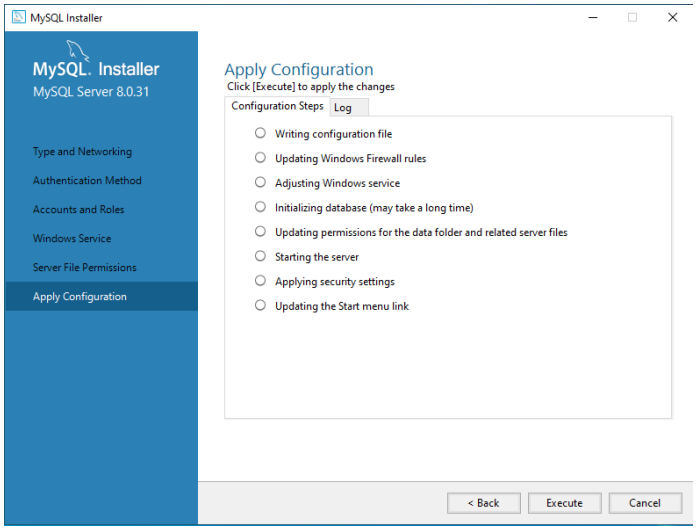


Click on **Next**

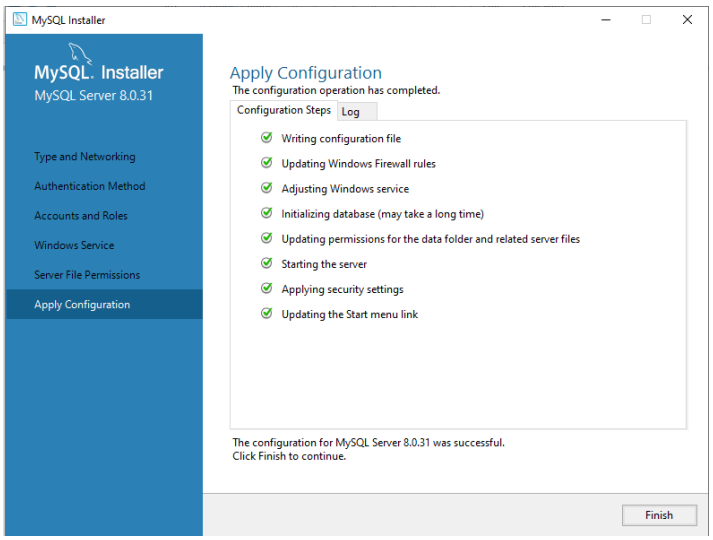


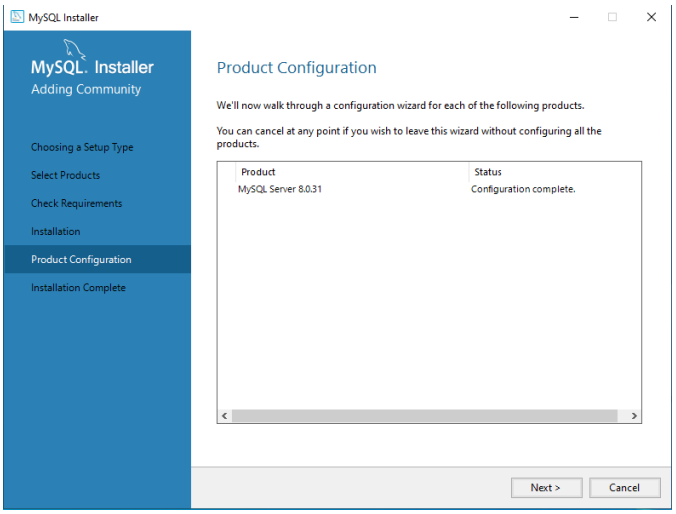
Click on **Next**

The next screen applies the configuration. Execute it. This step also takes a while to be concluded.



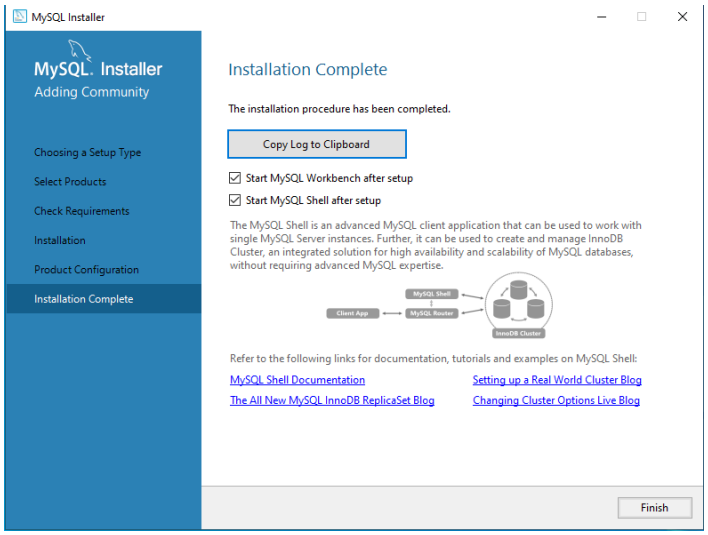
After it ends, just finish the process.



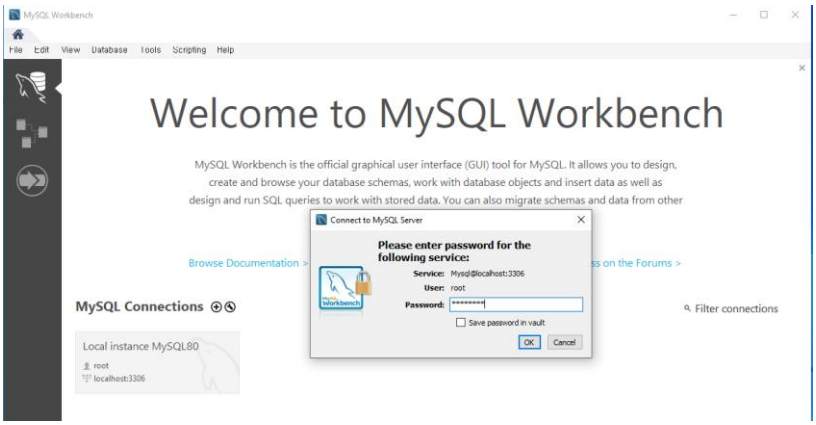


Click on **Next**

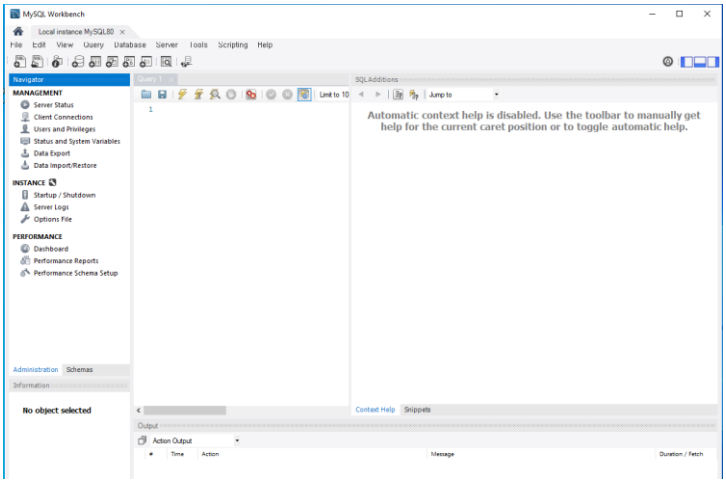
This screen is followed by another one asking to apply the configuration. Just execute it and click **Finish**.
We have finally reached the last screen.



Creating your first Database using MySQL Workbench
If you chose to start the Workbench after finishing the installation, you'll see the following screen:

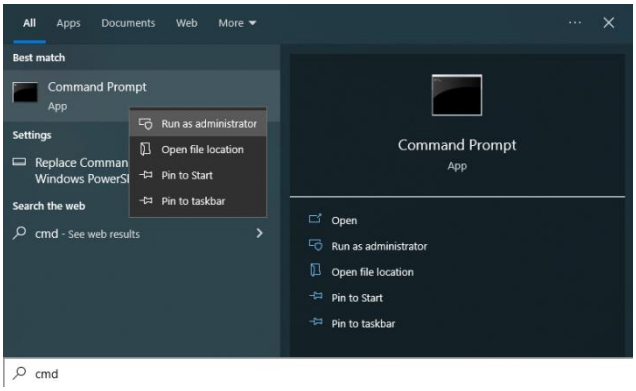


Choose the connection to the server you created and log into it.
- Input your **password**
- Click on **OK**
This is your working space:



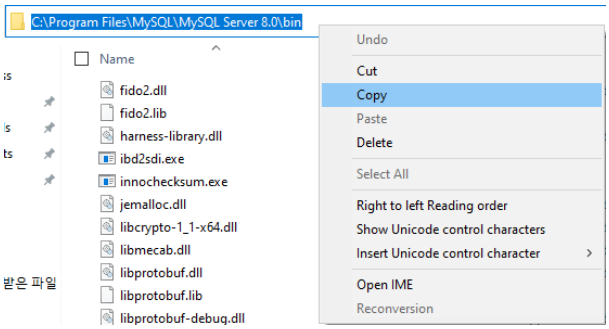
Notice in the **SCHEMAS** window that you already have a few sample databases to play with. In the **Information** window, you can see the database you have selected. Of course, you have the main window to write SQL code.

- Run the command prompt window in administrator mode. (If you want show database by command.)

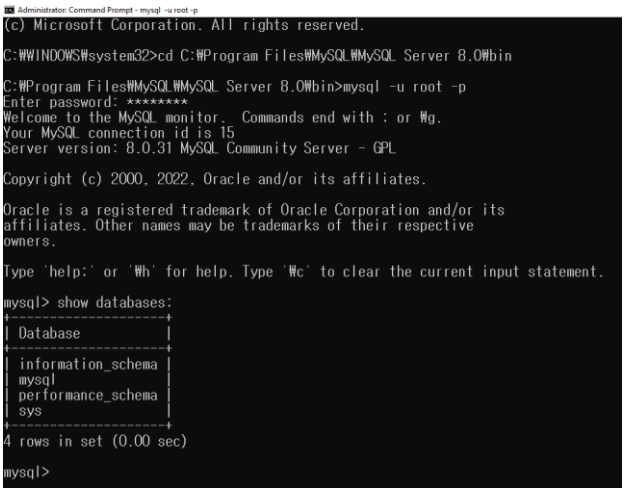


- Go to find path of MySQL Server 8.0bin

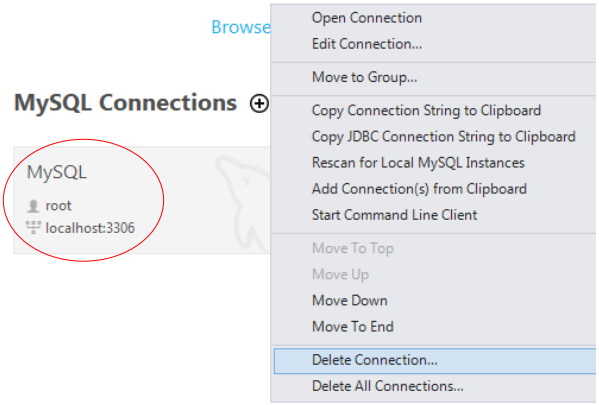
C:\Program Files\MySQL Server 8.0bin



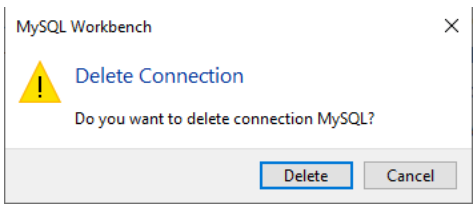
- cd C:\Program Files\MySQL Server 8.0bin
- mysql –u root –p
- password :……..
- mysql : show database;



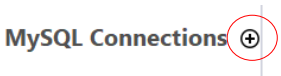
- Setup New connections



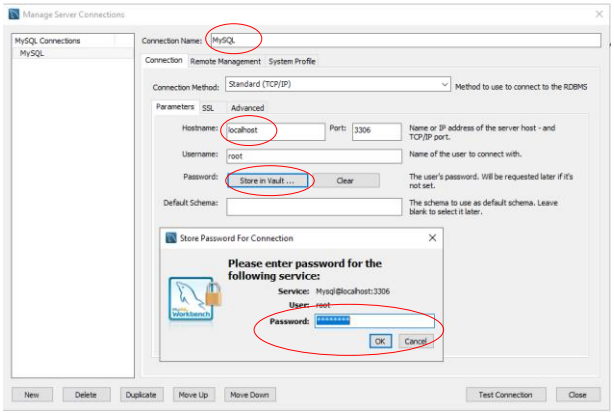
Right click on MySQL root and chose Delete Connection



Click on Delete



Click on MySQL Connections



- Connection Name: **MySQL**
- Hostname: **localhost**
- Click on Store in Vault…
- Password: input your password – Click on OK
- Click on Test Connection – Click on OK

• **DB #1: PHP Connect to MySQL**

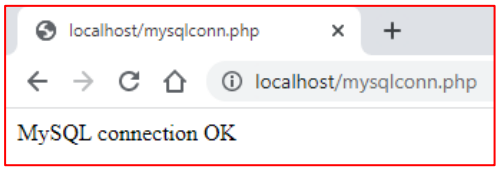
Open a **Connection to MySQL**

Before we can access data in the MySQL database, we need to be able to connect to the server:

1. Start **Apache**
2. Start **MySQL server**
3. Create new file name: **Mysqlconn.php** and **save** in **C:\Apache24\htdocs**

```
<?php
$host = 'localhost';
$user = 'root';
$pw = '';
$dbName = '';
$conn = new mysqli($host, $user, $pw, $dbName);
//Check SQL Command
if($conn){
   echo "MySQL connection OK";
}else{
   echo "MySQL connection failed";
}
?>
```

- The will produce the following Result:



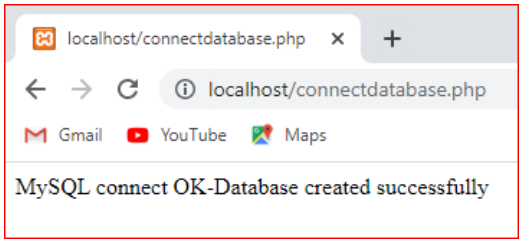
• DB #2: PHP Create a MySQL Database

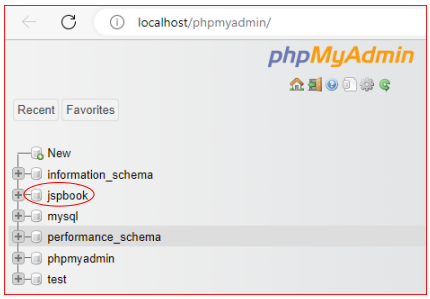
A database consists of one or more tables.

You will need special CREATE privileges to create or to delete a MySQL database.

```
$sql = "CREATE DATABASE jspbook";

//Check SQL Command
$sql = "CREATE DATABASE jspbook";
if ($conn->query($sql) === TRUE) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . $conn->error;
}

$conn->close();
```

- The will produce the following Result:





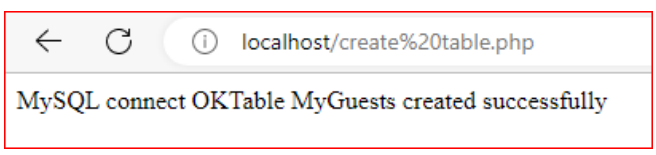
• DB #3: PHP MySQL Create Table

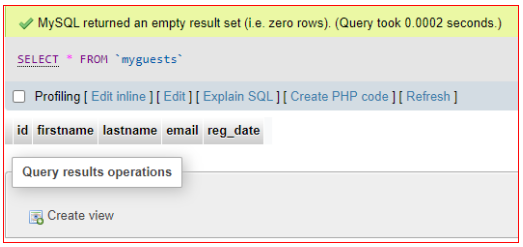
A database table has its own unique name and consists of columns and rows.

```
<?php
$host = 'localhost';
$user = 'root';
$pw = '';
```

```
$dbName = 'jspbook';
$conn = new mysqli($host, $user, $pw, $dbName);
//Check SQL Command
if($conn){
   echo "MySQL connect OK";
}else{
   echo "MySQL connection failed";
}
// sql to create table
$sql = "CREATE TABLE MyGuests (
   id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   firstname VARCHAR(30) NOT NULL,
   lastname VARCHAR(30) NOT NULL,
   email VARCHAR(50),
    reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
   CURRENT_TIMESTAMP
   )";

   if ($conn->query($sql) === TRUE) {
     echo  "Table MyGuests created successfully";
   } else {
     echo "Error creating table: " . $conn->error;
   }

   $conn->close();
?>
```

- The will produce the following Result:



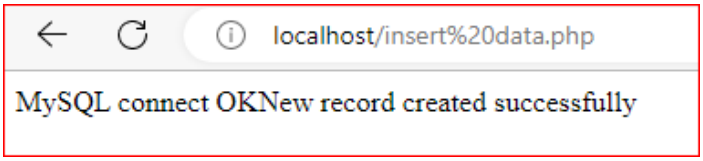


• DB #4: PHP MySQL Insert Data

After a database and a table have been created, we can start adding data in them.

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Chan', 'Dara', 'chandara168@gmail.com')";

if ($conn->query($sql) === TRUE) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
```

- The will produce the following Result:



• DB #5: PHP MySQL Get Last ID

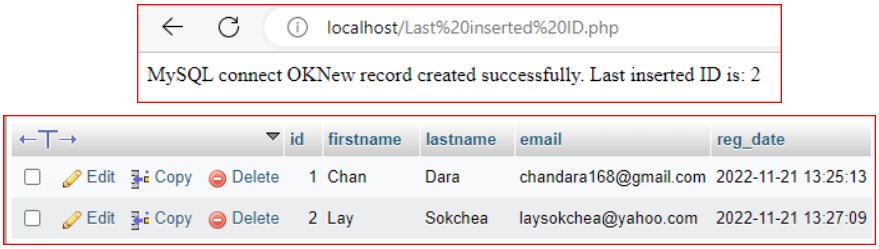
Get ID of The Last Inserted Record

If we perform an INSERT or UPDATE on a table with an AUTO_INCREMENT field, we can get the ID of the last inserted/updated record immediately.

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Lay', 'Sokchea', 'laysokchea@yahoo.com')";

if ($conn->query($sql) === TRUE) {
  $last_id = $conn->insert_id;
  echo "New record created successfully. Last inserted ID is: " . $last_id;
} else {
```

```
  echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
```

- The will produce the following Result:



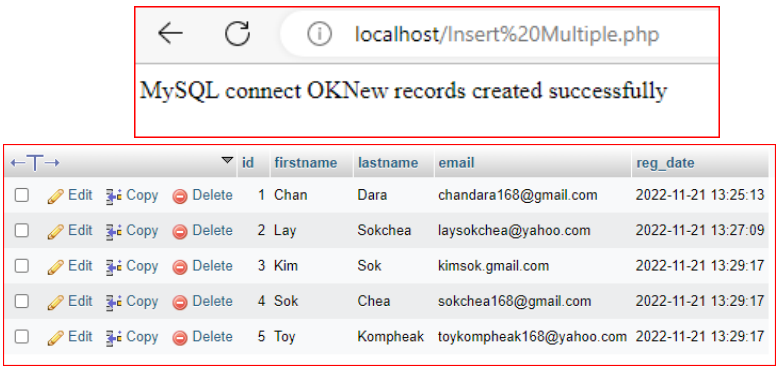
• DB #6: PHP MySQL Insert Multiple Records

Insert Multiple Records Into MySQL Using MySQLi

Multiple SQL statements must be executed with the mysqli_multi_query() function.

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Kim', 'Sok', 'kimsok.gmail.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Sok', 'Chea', 'sokchea168@gmail.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Toy', 'Kompheak', 'toykompheak168@yahoo.com')";

if ($conn->multi_query($sql) === TRUE) {
  echo "New records created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

- The will produce the following Result:

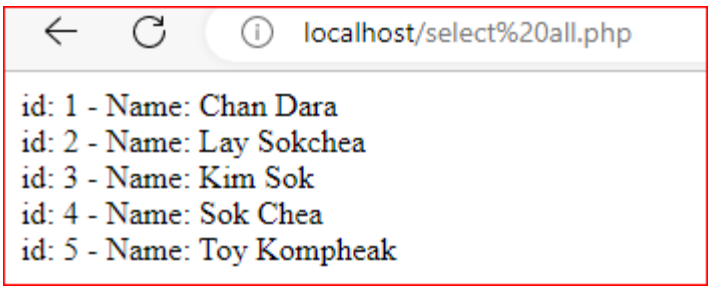


• DB #7: PHP MySQL Select Data

Select Data From a MySQL Database

```
$sql = "SELECT id, firstname, lastname, email FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
$row["lastname"]. "<br>";
  }
} else {
  echo "0 results";
}
$conn->close();
```

- The will produce the following Result:



• DB #8: PHP MySQL Use The WHERE Clause

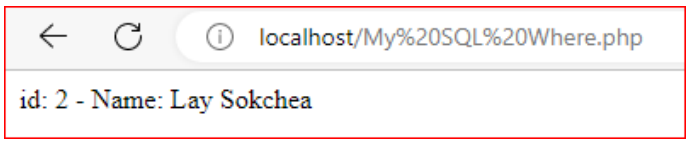
Select and Filter Data From a MySQL Database

The WHERE clause is used to filter records.

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

```
$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE
lastname='Sokchea'";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
$row["lastname"]. "<br>";
  }
} else {
  echo "0 results";
}
$conn->close();
```

- The will produce the following Result:



• DB #9: PHP MySQL Use The ORDER BY Clause

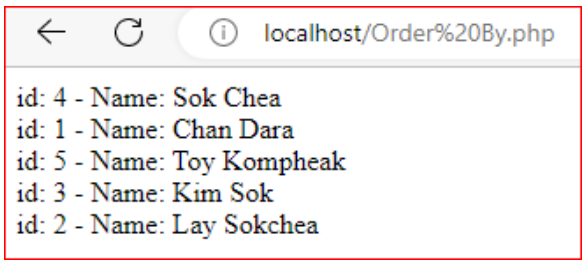
The ORDER BY clause is used to sort the result-set in ascending or descending order.

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

```
$sql = "SELECT id, firstname, lastname FROM MyGuests ORDER BY
lastname";
$result = $conn->query($sql);
```

```
if ($result->num_rows > 0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
            echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
$row["lastname"]. "<br>";
  }
} else {
  echo "0 results";
}
$conn->close();
```

- The will produce the following Result:

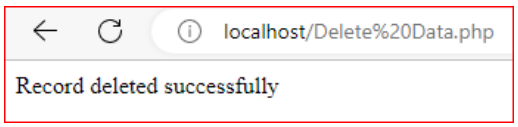


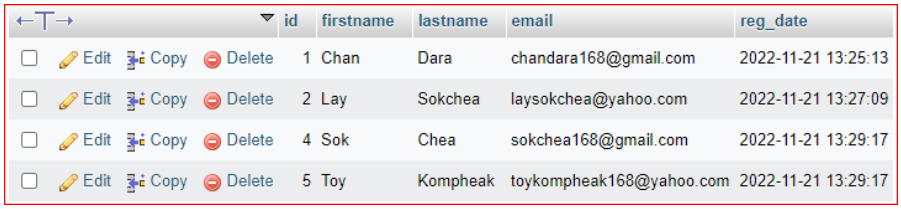
• DB #10: PHP MySQL Delete Data

The DELETE statement is used to delete records from a table:

```
$sql = "DELETE FROM MyGuests WHERE id=3";

if ($conn->query($sql) === TRUE) {
  echo "Record deleted successfully";
} else {
  echo "Error deleting record: " . $conn->error;
}
$conn->close();
```

- The will produce the following Result:





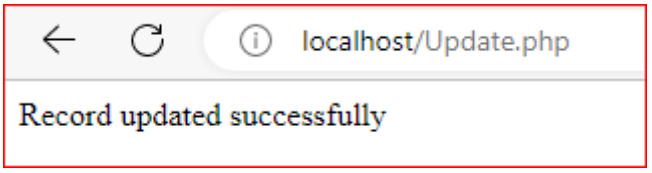
	id	firstname	lastname	email	reg_date
Edit Copy Delete	1	Chan	Dara	chandara168@gmail.com	2022-11-21 13:25:13
Edit Copy Delete	2	Lay	Sokchea	laysokchea@yahoo.com	2022-11-21 13:27:09
Edit Copy Delete	4	Sok	Chea	sokchea168@gmail.com	2022-11-21 13:29:17
Edit Copy Delete	5	Toy	Kompheak	toykompheak168@yahoo.com	2022-11-21 13:29:17

• DB #11: PHP MySQL Update Data

The UPDATE statement is used to update existing records in a table:

```
$sql = "UPDATE MyGuests SET lastname='Rithy' WHERE id=2";

if ($conn->query($sql) === TRUE) {
  echo "Record updated successfully";
} else {
  echo "Error updating record: " . $conn->error;
}

$conn->close();
```

- The will produce the following Result:



Summary

- *Apache is the most commonly used Web server on Linux systems. Web servers are used to serve Web pages requested by client computers. Clients typically request and view Web pages using Web browser applications such as Firefox, Opera, Chromium, or Internet Explorer.*
- *PHP (Hypertext Preprocessor) is known as a general-purpose scripting language that can be used to develop dynamic and interactive websites. It was among the first server-side languages that could be embedded into HTML, making it easier to add functionality to web pages without needing to call external files for data.*
- *MySQL is a tool used to manage databases and servers, so while it's not a database, it's widely used in relation to managing and organising data in databases.*

Questions

1. Why we need install Apache server?
2. What does it mean to install PHP?
3. What is MySQL How to you Install?
4. How can connect to MySQL database in PHP?

Exercises

1. Please write code connect to MySQL database called rttcDB in PHP.
2. Please write code create database called rttcDB in PHP.
3. Please write code to create table grade in PHP follow table below:

Field	*Type*	*Null*	*Key*	*Default*
ID	*Integer(3)*	*Yes*	*Primary Key*	*Null*
khmer	*Integer(3)*	*Yes*		*Null*
math	*Integer(3)*	*Yes*		*Null*
science	*Var(3)*	*Yes*		*Null*

4. Please write code to insert 3 students into table grade in PHP follow table below:

ID	*khmer*	*math*	*science*	*comment*
111	*90*	*95*	*85*	*outstanding*
112	*100*	*95*	*75*	*kind warm*
113	*70*	*90*	*80*	*need to study hard*

5. Please write code update table grade set math=60 with Id 112 in PHP.
6. Please write code select all from table grade in PHP.

Chapter 07

Data and Process

Data is digitalized and stored with various facts, making it the most important material to understand and solve problems. Various data analysis tools can be used to improve students' computing thinking skills, and this chapter uses a tool called Orange 3. Chapter 7 data and processing learn various ways to load data as a basic step to analyze data using Orange 3. In addition, to check the data in detail, it is expressed in a table form and columns and rows can be selected and utilized according to conditions. Finally, to analyze the data, learn how to preprocess it into the most suitable form and deal with the data yourself.

In this chapter, you will learn:

7.1. File

Reads attribute-value data from an input file.

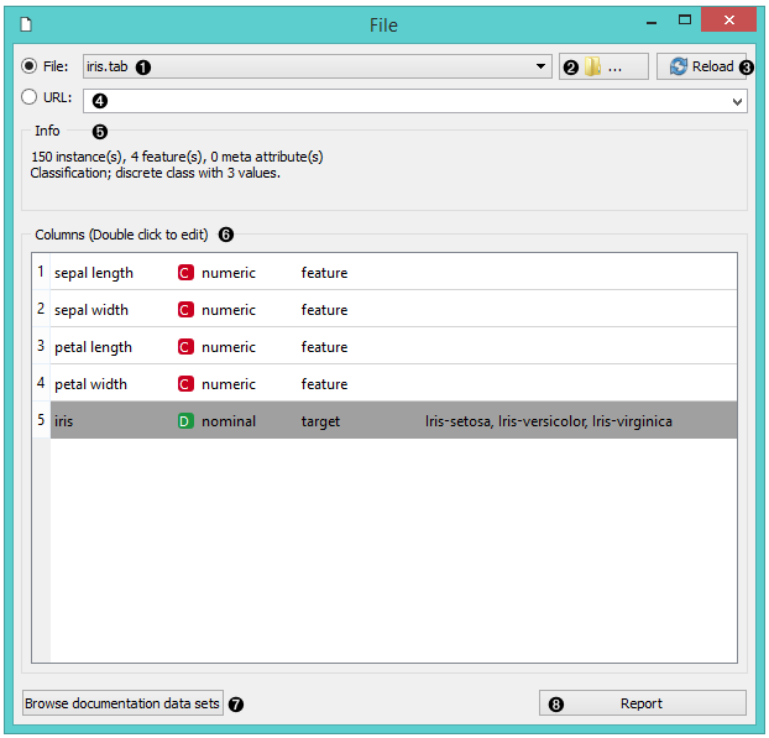
7.1.1. Output

- Data: dataset from file.

The File widget reads the input data file (data table with data instances) and sends the dataset to its output channel. The history of the most recently opened files is maintained in the widget. The widget also includes a directory with sample datasets that come pre-installed with Orange.

The widget reads data from Excel (.xlsx), simple tab-delimited (.txt), comma-separated files (.csv) or URLs. For other formats see the Other Formats section below.

Figure 7.1 File



1. Browse through previously opened data files, or load any of the sample ones.
2. Browse for a data file.
3. Reloads the currently selected data file.
4. Insert data from URL addresses, including data from Google Sheets.
5. Information on the loaded dataset: dataset size, number, and types of data features.

6. Additional information on the features in the dataset. Features can be edited by double-clicking on them. The user can change the attribute names, select the type of variable per each attribute (Continuous, Nominal, String, Datetime), and choose how to further define the attributes (as Features, Targets or Meta). The user can also decide to ignore an attribute.
7. Browse documentation datasets.
8. Produce a report.

Example:

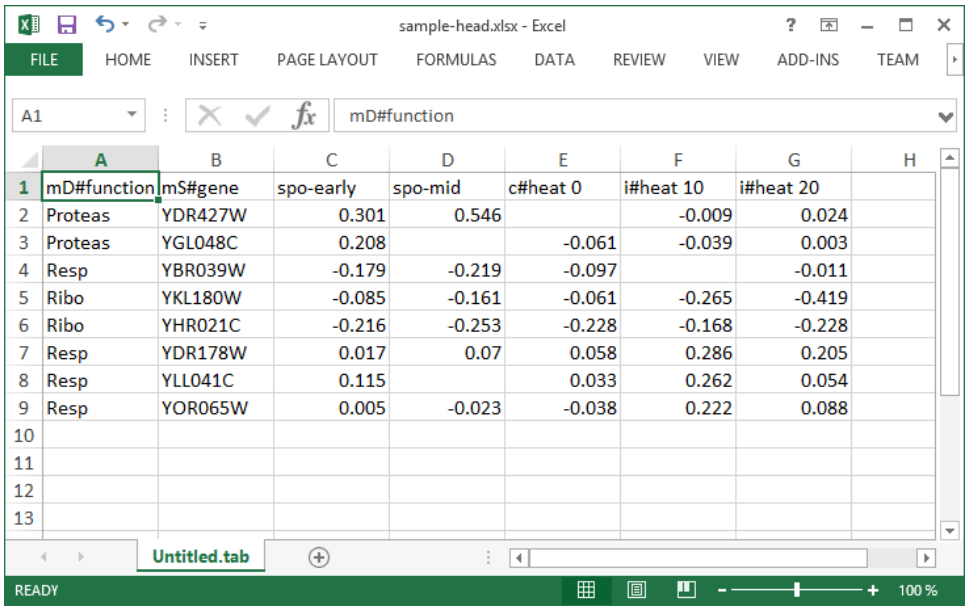
Most Orange workflows would probably start with the File widget. In the schema below, the widget is used to read the data that is sent to both the Data Table and the Box Plot widget.

Figure 7.2 File Link to Data and Box plot Table

7.1.2. Loading your data

1. Orange can import any comma, .xlsx, or tab-delimited data file or URL. Use the File widget and then, if needed, select class and meta attributes.
2. To specify the domain and the type of the attribute, attribute names can be preceded by a label followed by a hash. Use c for class and m for meta attribute, I to ignore a column, and C, D, S for continuous, discrete and string attribute types. Examples: C#mpg, mS#name, i#dummy.
3. Orange's native format is a tab-delimited text file with three header rows. The first row contains attribute names, the second the type (continuous, discrete or string), and the third the optional element (class, meta or time).

Figure 7.3 Sample Format Text File



	A	B	C	D	E	F	G	H
1	mD#function	mS#gene	spo-early	spo-mid	c#heat 0	i#heat 10	i#heat 20	
2	Proteas	YDR427W	0.301	0.546		-0.009	0.024	
3	Proteas	YGL048C	0.208		-0.061	-0.039	0.003	
4	Resp	YBR039W	-0.179	-0.219	-0.097		-0.011	
5	Ribo	YKL180W	-0.085	-0.161	-0.061	-0.265	-0.419	
6	Ribo	YHR021C	-0.216	-0.253	-0.228	-0.168	-0.228	
7	Resp	YDR178W	0.017	0.07	0.058	0.286	0.205	
8	Resp	YLL041C	0.115		0.033	0.262	0.054	
9	Resp	YOR065W	0.005	-0.023	-0.038	0.222	0.088	
10								
11								
12								
13								

Read more on loading your data here.

7.1.3. Other Formats

Supported formats and the widgets to load them:

- distance matrix: Distance File
- predictive model: Load Model
- network: Network File from Network add-on
- images: Import Images from the Image Analytics add-on
- text/corpus: Corpus or Import Documents from Text add-on
- single-cell data: Load Data from Single Cell add-on
- several spectroscopy files: Multifile from Spectroscopy add-on

7.2. CSV File Import

Import a data table from CSV formatted file.

7.2.1. Output

- Data: dataset from the .csv file
- Data Frame: pandas DataFrame object

The CSV File Import widget reads comma-separated files and sends the dataset to its output channel. File separators can be commas, semicolons, spaces, tabs, or manually-defined delimiters. The history of the most recently opened files is maintained in the widget.

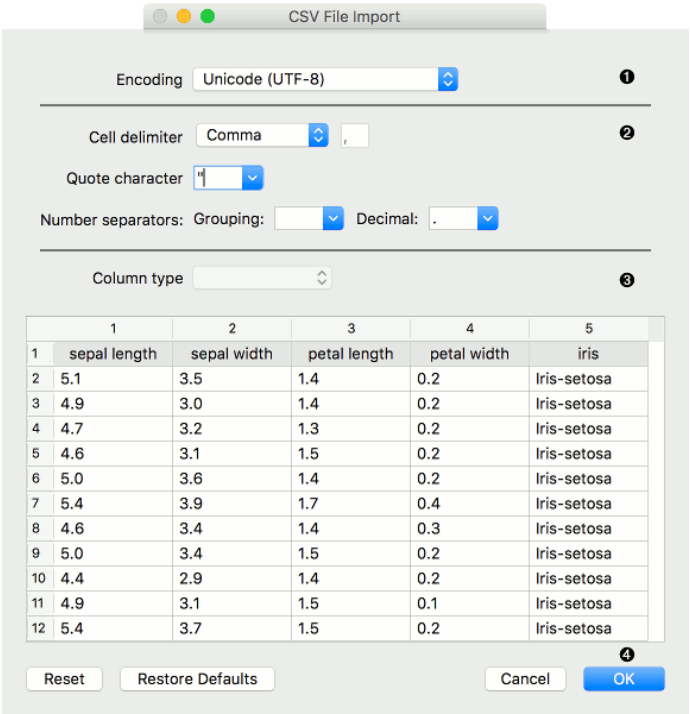
Data Frame output can be used in the Python Script widget by connecting it to the in-object input (e.g. df = in object). Then it can be used as a regular DataFrame.

7.2.2. Import Options

The import window is where the user sets the import parameters. Can be re-opened by pressing Import Options in the widget.

Right-click on the column name to set the column type. Right-click on the row index (on the left) to mark a row as a header, skipped or a normal data row.

Figure 7.4 CSV File Import



	1	2	3	4	5
1	sepal length	sepal width	petal length	petal width	iris
2	5.1	3.5	1.4	0.2	Iris-setosa
3	4.9	3.0	1.4	0.2	Iris-setosa
4	4.7	3.2	1.3	0.2	Iris-setosa
5	4.6	3.1	1.5	0.2	Iris-setosa
6	5.0	3.6	1.4	0.2	Iris-setosa
7	5.4	3.9	1.7	0.4	Iris-setosa
8	4.6	3.4	1.4	0.3	Iris-setosa
9	5.0	3.4	1.5	0.2	Iris-setosa
10	4.4	2.9	1.4	0.2	Iris-setosa
11	4.9	3.1	1.5	0.1	Iris-setosa
12	5.4	3.7	1.5	0.2	Iris-setosa

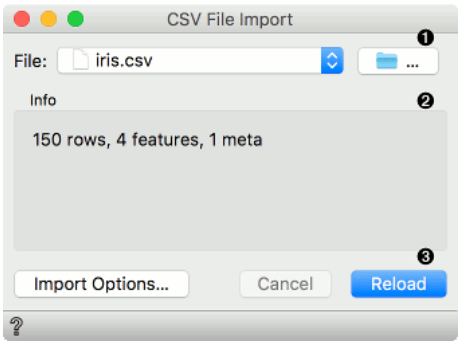
1. File encoding. The default is UTF-8. See the Encoding subchapter for details.
2. Import settings:
   - Cell delimiter:
     - Tab
     - Comma
     - Semicolon

  - Space
  - Other (set the delimiter in the field to the right)
- Quote character: either “ or ‘. Defines what is considered a text.
- Number separators:
  - Grouping: delimiters for thousands, e.g. 1,000
  - Decimal: delimiters for decimals, e.g. 1.234

3. Column type: select the column in the preview and set its type. Column type can be set also by right-clicking on the selected column.
   - Auto: Orange will automatically try to determine column type. (default)
   - Numeric: for continuous data types, e.g. (1.23, 1.32, 1.42, 1.32)
   - Categorical: for discrete data types, e.g. (brown, green, blue)
   - Text: for string data types, e.g. (John, Olivia, Mike, Jane)
   - Datetime: for time variables, e.g. (1970-01-01)
   - Ignore: do not output the column.
4. Pressing Reset will return the settings to the previously set state (saved by pressing OK in the Import Options dialogue). Restore Defaults will set the settings to their default values. Cancel aborts the import, while OK imports the data and saves the settings.

7.2.3. Widget

The widget once the data is successfully imported.



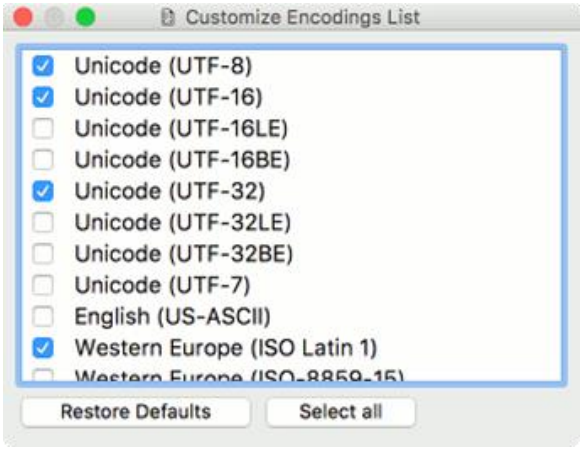
1. The folder icon opens the dialogue for importing the local .csv file. It can be used to either load the first file or change the existing file (load new data). The File dropdown stores paths to previously loaded data sets.
2. Information on the imported data set. Reports on the number of instances (rows), variables (features or columns), and meta-variables (special columns).

3. Import Options re-opens the import dialogue where the user can set delimiters, encodings, text fields, and so on. Cancel aborts data import. Reload imports the file once again, adding to the data any changes made in the original file.

7.2.4. Encoding

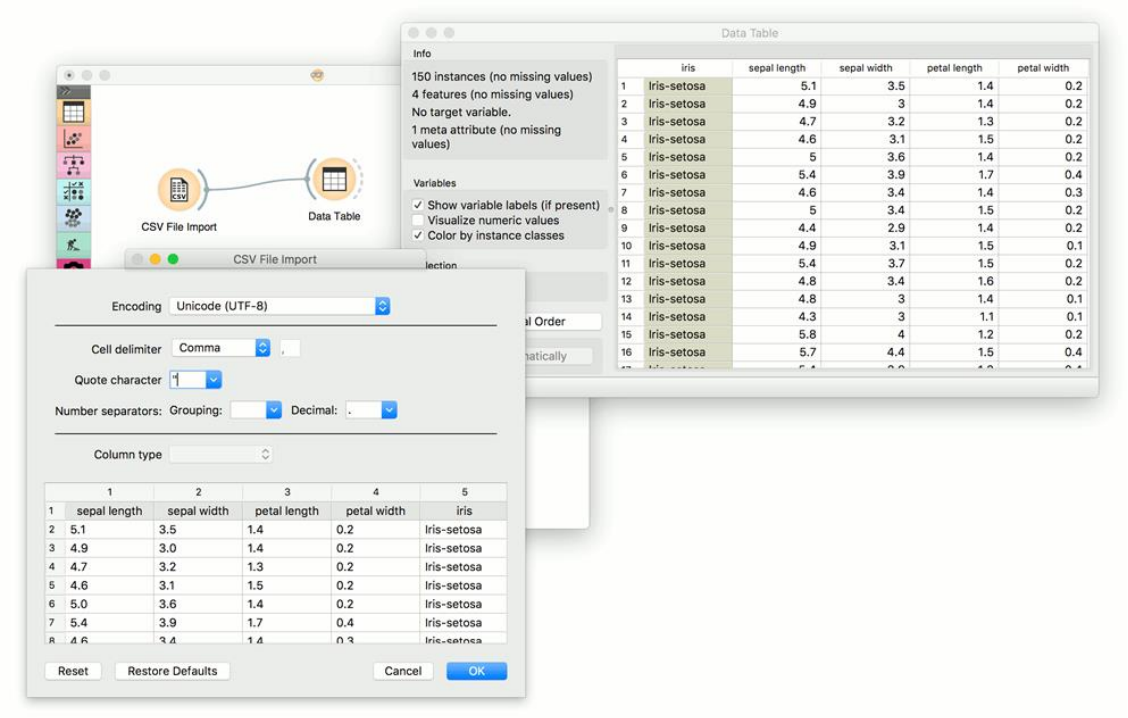
The dialogue for settings custom encodings list in the Import Options - Encoding dropdown. Select Customize Encodings List. . . to change which encodings appear in the list. To save the changes, simply close the dialogue. Closing and reopening Orange (even with Reset widget settings) will not reset the list. To do this, press Restore Defaults. To have all the available encodings in the list, press Select all.

Figure 7.5 Customize Encoding List



Example:

CSV File Import works almost exactly like the File widget, with the added options for importing different types of .csv files. In this workflow, the widget read the data from the file and sends it to the Data Table for inspection.



7.3. Datasets

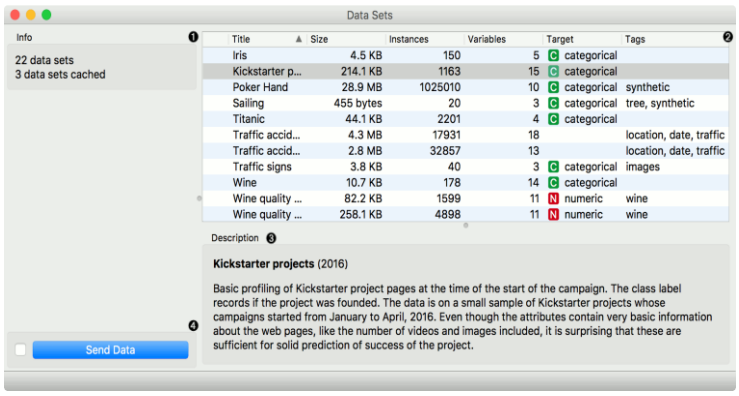
Load a dataset from online repository.

7.3.1. Output

- Data: output dataset

The datasets widget retrieves the selected dataset from the server and sends it to the output. The file is downloaded to the local memory and thus instantly available even without an internet connection. Each dataset is provided with a description and formation on the data size, number of instances, number of variables, target, and tags.

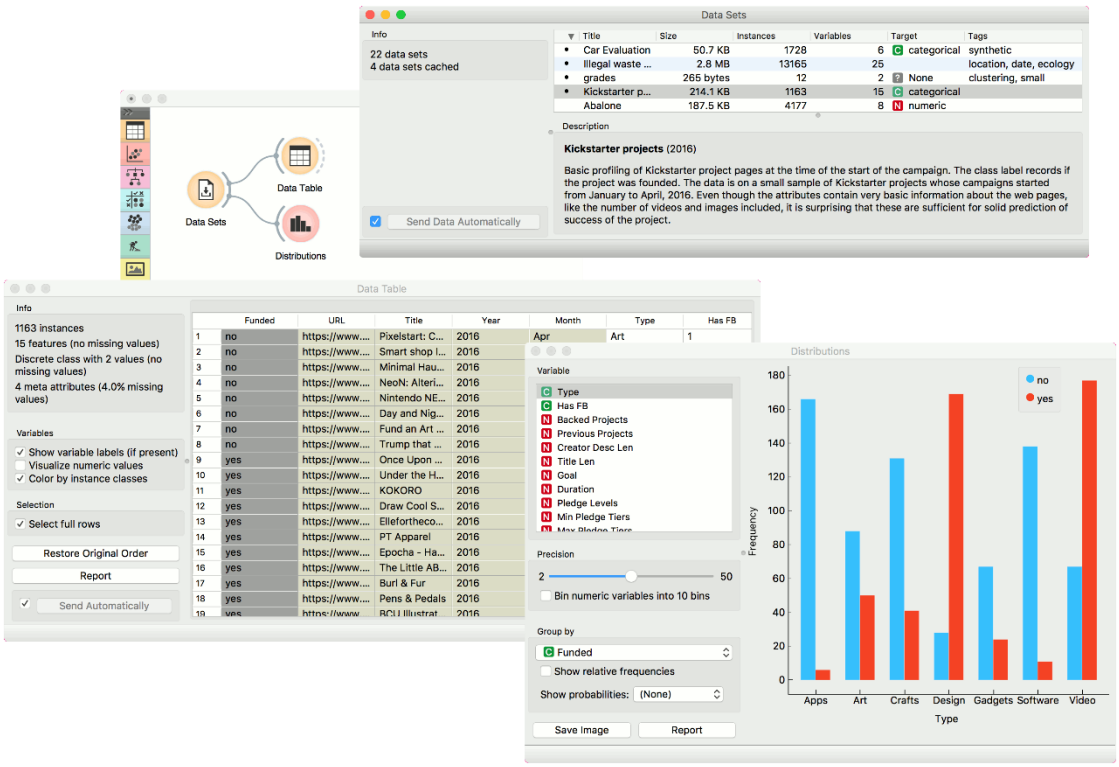
Figure 7.6 Dataset File



1. Information on the number of datasets available and the number of them downloaded to the local memory.
2. Content of available datasets. Each dataset is described with the size, the number of instances and variables, the type of the target variable, and tags.
3. Formal description of the selected dataset.
4. If Send Data Automatically is ticked, the selected dataset is communicated automatically. Alternatively, press Send Data.

Example:

Orange workflows can start with Datasets widget instead of the File widget. In the example below, the widget retrieves a dataset from an online repository (Kickstarter data), which is subsequently sent to both the Data Table and the Distributions.



7.4. Data Table

Displays attribute value data in a spreadsheet in table form.

7.4.1. Input

- Data: input dataset

7.4.2.Output

- Selected Data: instances selected from the table

The Data Table widget receives one or more datasets in its input and presents them as a spreadsheet. Data instances may be sorted by attribute values. The widget also supports manual selection of data instances.

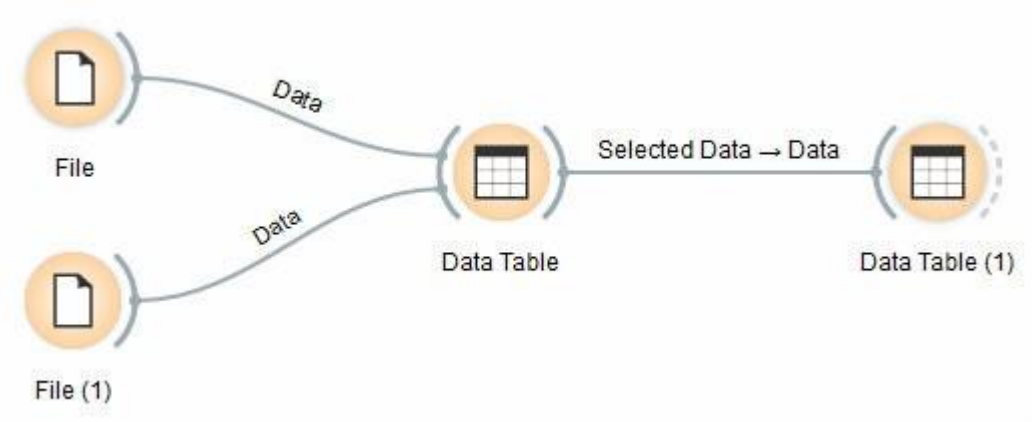
Figure 7.7 Data Table File

1. The name of the dataset (usually the input data file). Data instances are in rows and their attribute values in columns. In this example, the dataset is sorted by the attribute "sepal length".
2. Info on current dataset size and number and types of attributes
3. Values of continuous attributes can be visualized with bars; colors can be attributed to different classes.
4. Data instances (rows) can be selected and sent to the widget's output channel.
5. Use the Restore Original Order button to reorder data instances after attribute-based sorting.

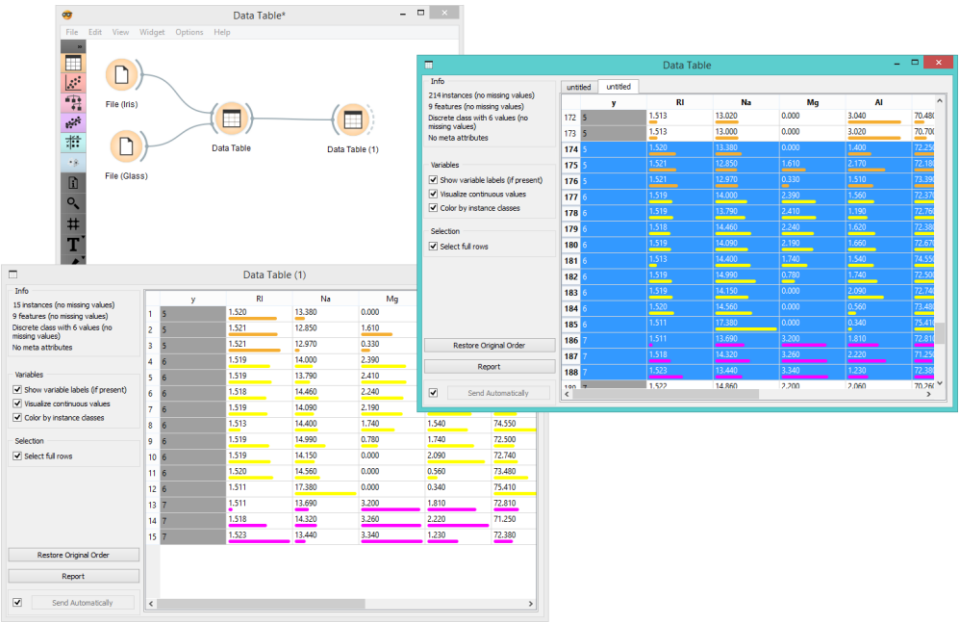
6. Produce a report.
7. While auto-send is on, all changes will be automatically communicated to other widgets. Otherwise, press Send Selected Rows.

Example:

We used two File widgets to read the Iris and Glass dataset (provided in the Orange distribution) and send them to the Data Table widget.



Selected data instances in the first Data Table are passed to the second Data Table. Notice that we can select which dataset to view (iris or glass). Changing from one dataset to another alters the communicated selection of data instances if Commit on any change is selected.



7.5. Select Columns

Manual selection of data attributes and composition of the data domain.

7.5.1. Input

- Data: input dataset

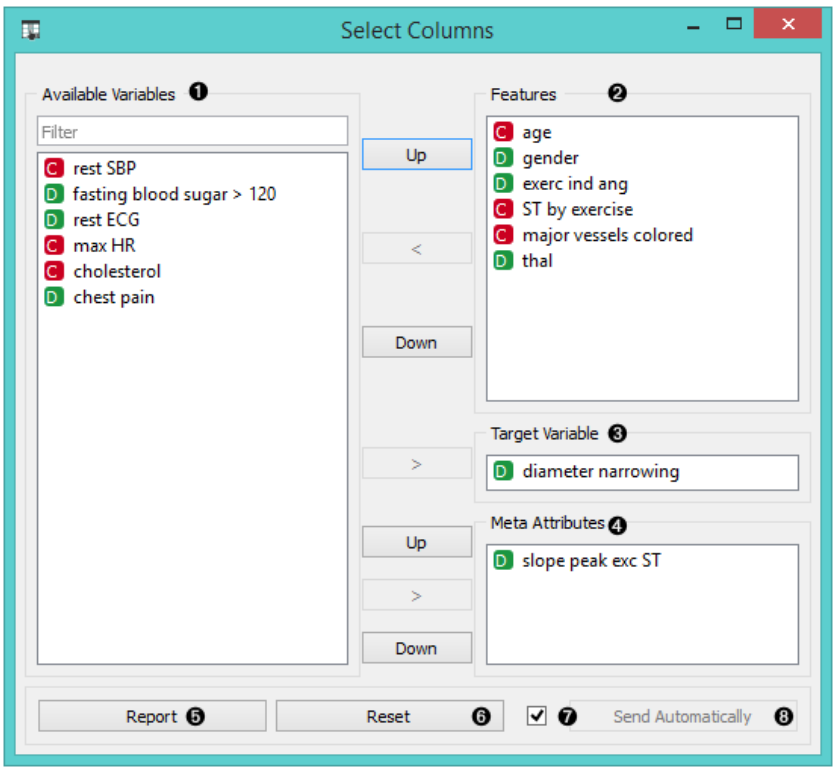
7.5.2. Outputs

- Data: dataset with columns as set in the widget.

The Select Columns widget is used to manually compose your data domain. The user can decide which attributes will be used and how. Orange distinguishes between ordinary attributes, (optional) class attributes, and meta-attributes. For instance, for building a classification model, the domain would be composed of a set of attributes and a discrete class attribute. Meta attributes are not used in modelling, but several widgets can use them as instance labels.

Orange attributes have a type and are either discrete, continuous or a character string. The attribute type is marked with a symbol appearing before the name of the attribute (D, C, S, respectively).

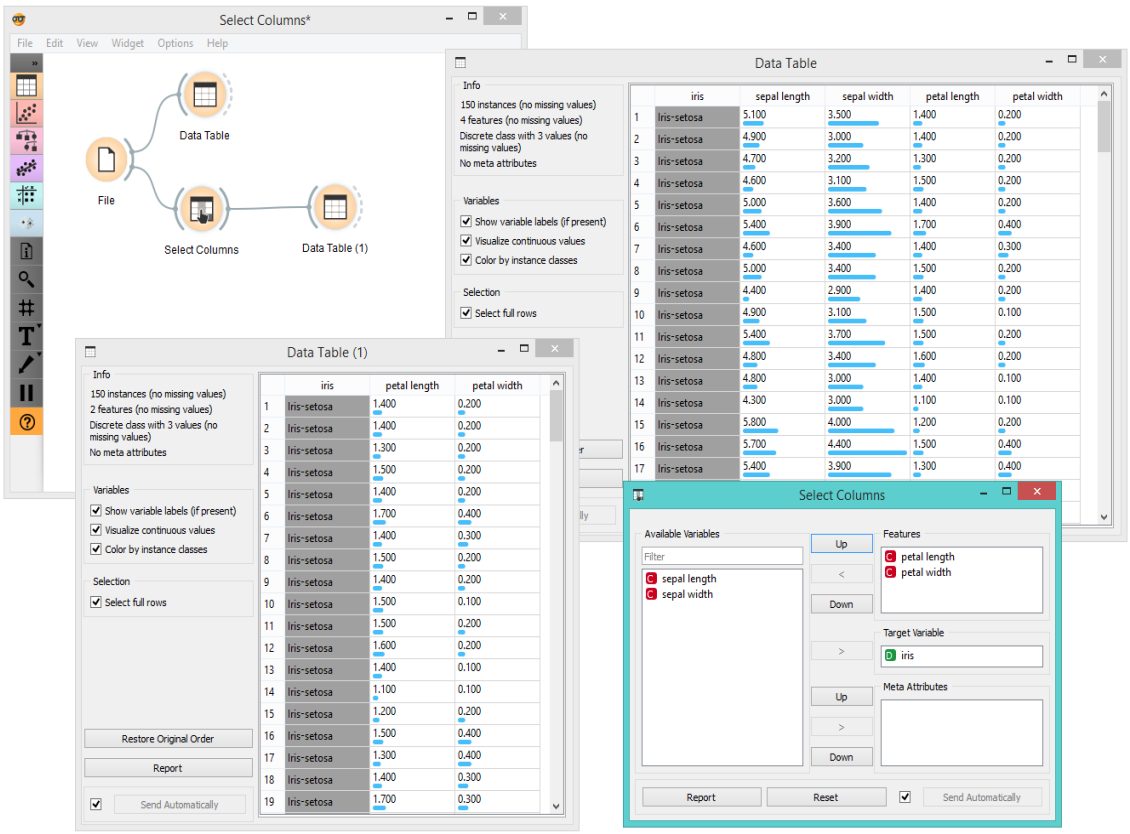
Figure 7.8 Select Columns



1. Left-out data attributes that will not be in the output data file
2. Data attributes in the new data file
3. Target variable. If none, the new dataset will be without a target variable.
4. Meta attributes of the new data file. These attributes are included in the dataset but are, for most methods, not considered in the analysis.
5. Produce a report.
6. Reset the domain composition to that of the input data file.
7. Tick if you wish to auto-apply changes of the data domain.
8. Apply changes of the data domain and send the new data file to the output channel of the widget.

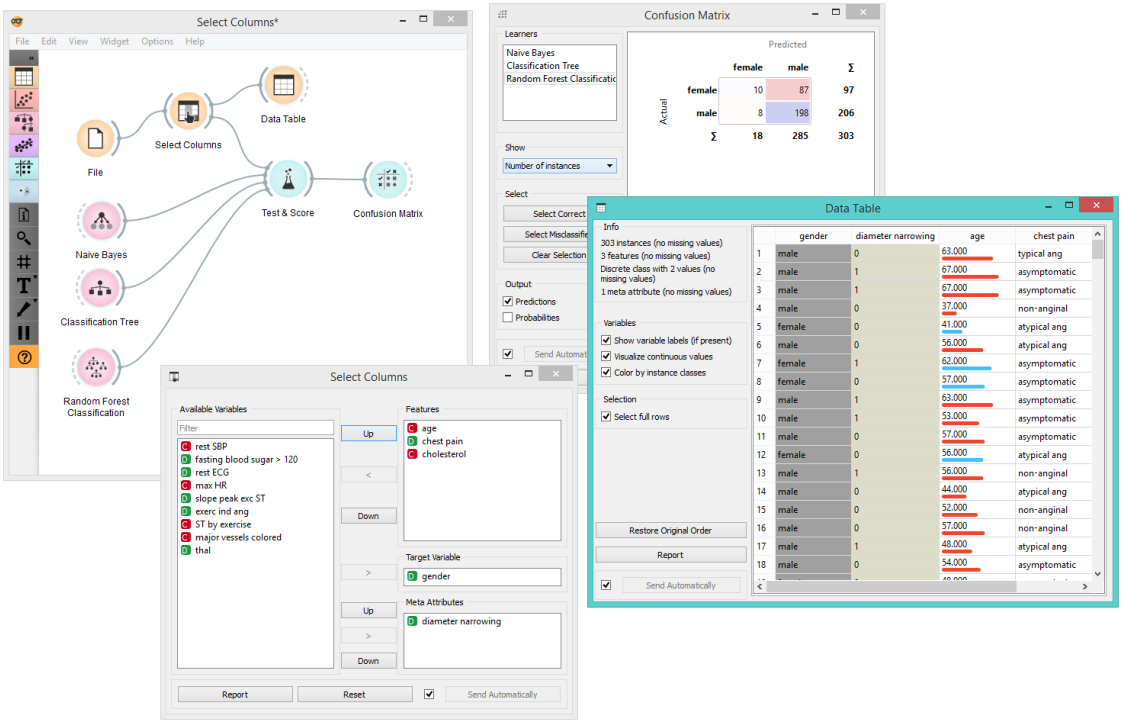
Example:

In the workflow below, the Iris data from the File widget is fed into the Select Columns widget, where we select to output only two attributes (namely petal width and petal length). We view both the original dataset and the dataset with selected columns in the Data Table widget.



For a more complex use of the widget, we composed a workflow to redefine the classification problem in the heart- disease dataset. Originally, the task was to predict if the patient has a coronary artery diameter narrowing. We changed the problem to that of gender classification, based on age, chest pain and cholesterol level, and informatively kept the diameter narrowing as a meta attribute.

Figure 7.9 Select Columns and Data Table File



7.6. Select Rows

Selects data instances based on conditions over data features.

7.6.1. Inputs

- Data: input dataset

7.6.1. Output

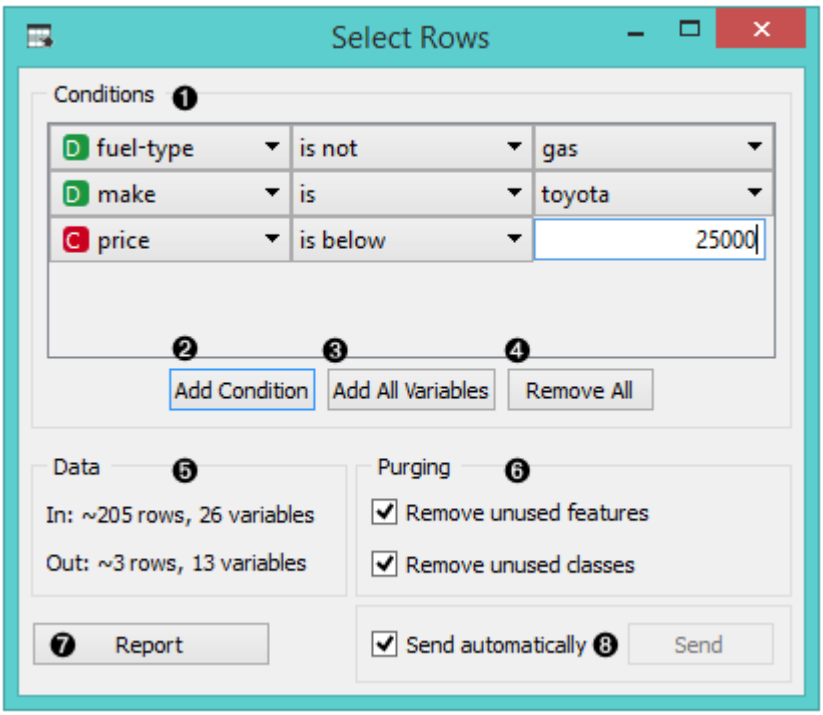
- Matching Data: instances that match the conditions
- Non-Matching Data: instances that do not match the conditions
- Data: data with an additional column showing whether a instance is selected

This widget selects a subset from an input dataset, based on user-defined conditions. Instances that match the selection rule are placed in the output Matching Data channel.

Criteria for data selection are presented as a collection of conjunct terms (i.e. selected items are those matching all the terms in ‘Conditions’).

Condition terms are defined by selecting an attribute, selecting an operator from a list of operators, and, if needed, defining the value to be used in the condition term. Operators are different for discrete, continuous and string attributes.

Figure 7.10 Select Rows



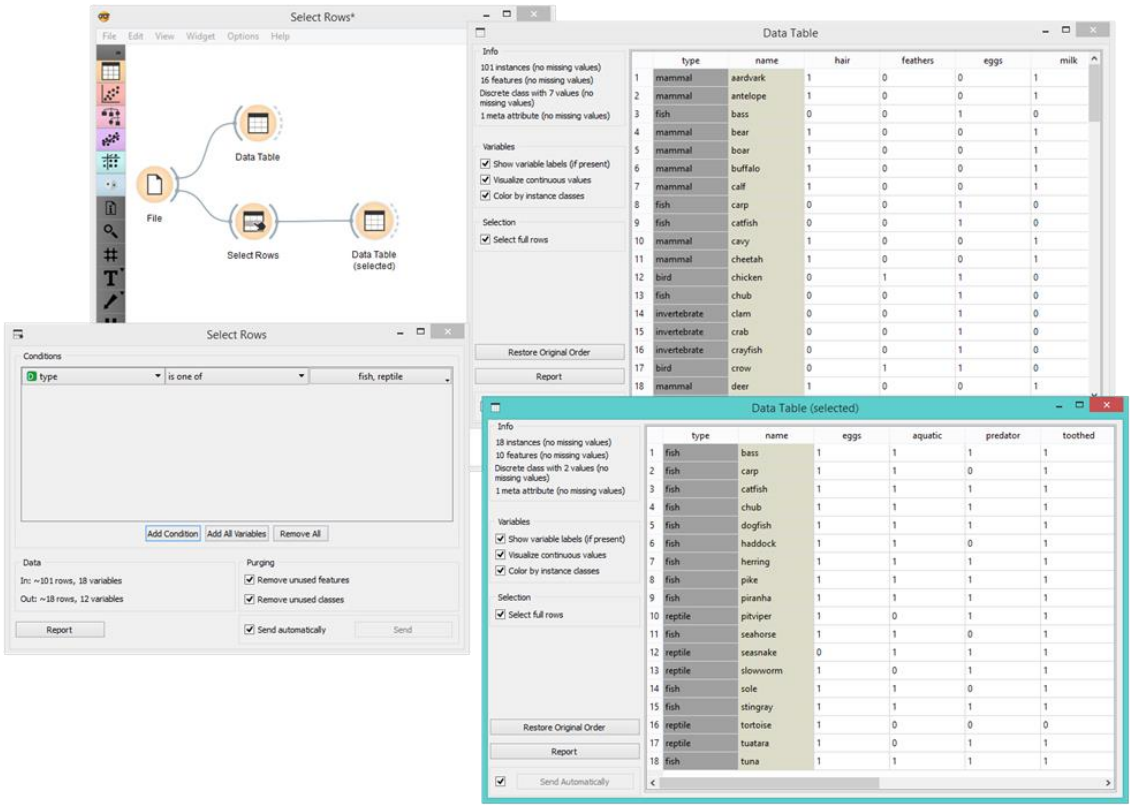
1. Conditions you want to apply, their operators and related values
2. Add a new condition to the list of conditions.
3. Add all the possible variables at once.
4. Remove all the listed variables at once.
5. Information on the input dataset and information on instances that match the condition(s)
6. Purge the output data.
7. When the Send automatically box is ticked, all changes will be automatically communicated to other widgets.
8. Produce a report.

Any change in the composition of the condition will update the information pane (Data Out).

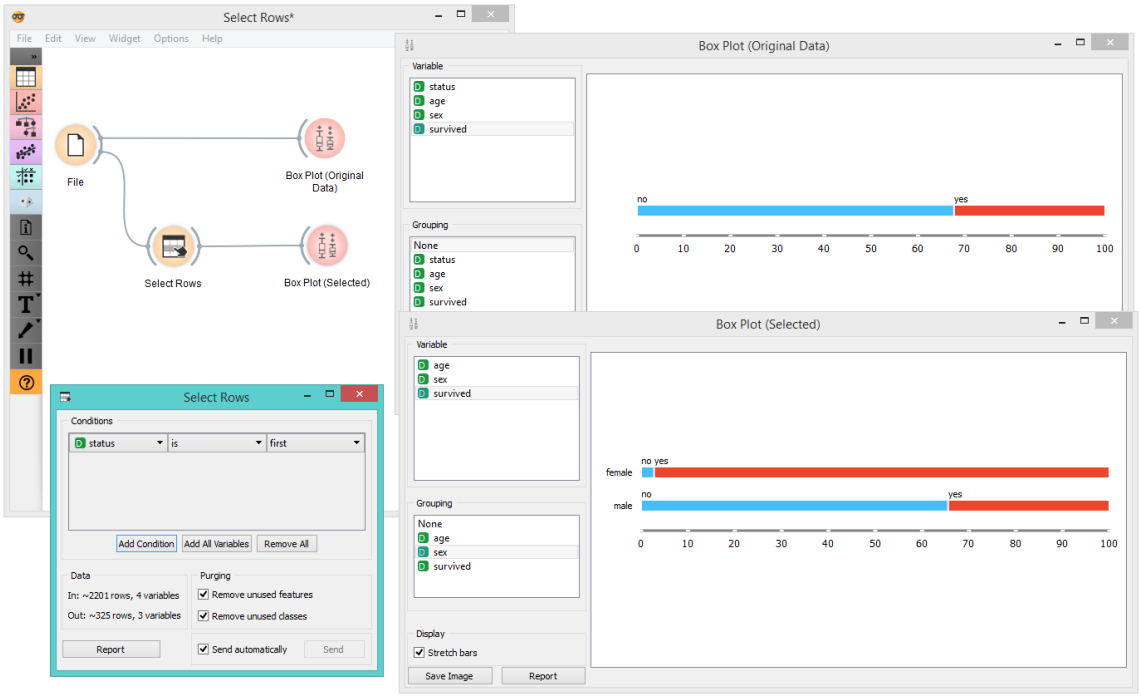
If Send automatically is selected, then the output is updated on any change in the composition of the condition or any of its terms.

Example:

In the workflow below, we used the Zoo data from the File widget and fed it into the Select Rows widget. In the widget, we chose to output only two animal types, namely fish and reptiles. We can inspect both the original dataset and the dataset with selected rows in the Data Table widget.



In the next example, we used the data from the Titanic dataset and similarly fed it into the Box Plot widget. We first observed the entire dataset based on survival. Then we selected only first class passengers in the Select Rows widget and fed it again into the Box Plot. There we could see all the first class passengers listed by their survival rate and grouped by gender.



7.7. Impute

Replaces unknown (missing)values in the data.

7.7.1. Inputs

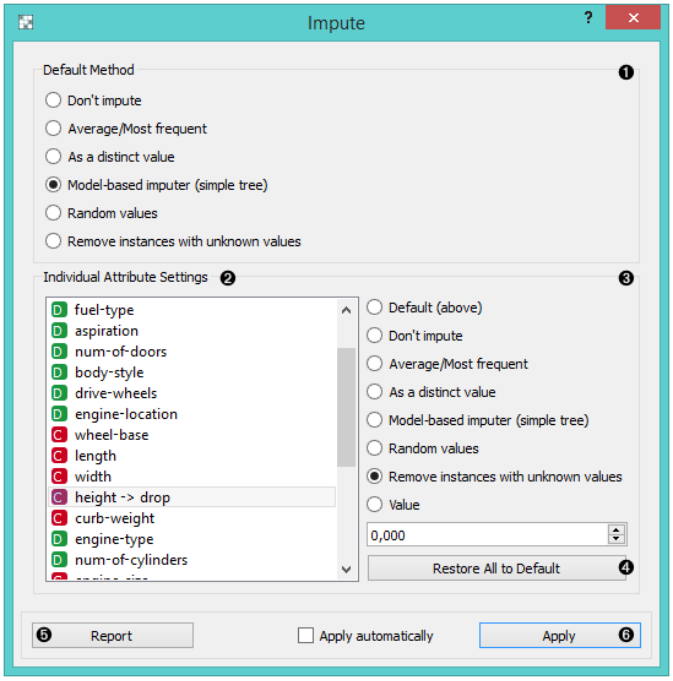
- Data: input dataset
- Learner: learning algorithm for imputation

7.7.2. Outputs

- Data: dataset with imputed values

Some of Orange's algorithms and visualizations cannot handle unknown values in the data. This widget does what statisticians call imputation: it substitutes missing values by values either computed from the data or set by the user. The default imputation is (1-NN).

Figure 7.11 Impute File



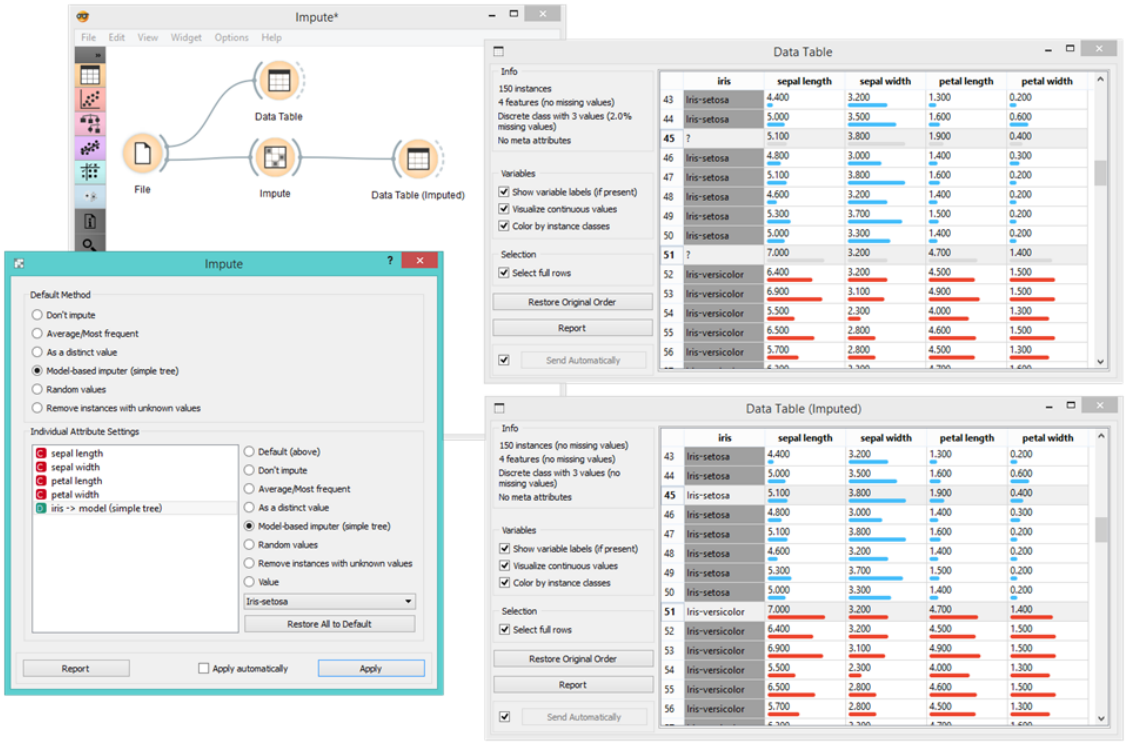
1. In the top-most box, Default method, the user can specify a general imputation technique for all attributes.
   - **Don't Impute** does nothing with the missing values.
   - **Average/Most-frequent** uses the average value (for continuous attributes) or the most common value (for discrete attributes).
   - **As a distinct value** creates new values to substitute the missing ones.
   - **Model-based imputer** constructs a model for predicting the missing value, based on values of other at- tributes; a separate model is constructed for each attribute. The default model is 1-NN learner, which takes the value from the most similar example (this is sometimes referred to as hot deck imputation). This al- gorithm can be substituted by one that the user connects to the input signal Learner for Imputation. Note, however, that if there are discrete and continuous attributes in the data, the algorithm needs to be capable of handling them both; at the moment only 1-NN learner can do that. (In the future, when Orange has more regressors, the Impute widget may have separate input signals for discrete and continuous models.)
   - **Random values** compute the distributions of values for each attribute and then impute by picking random values from them.

- **Remove examples with missing values** and removes the example containing missing values. This check also applies to the class attribute if Impute class values is checked.

2. It is possible to specify individual treatment for each attribute, which overrides the default treatment set. One can also specify a manually defined value used for imputation. In the screenshot, we decided not to impute the values of “normalized-losses” and “make”, the missing values of “aspiration” will be replaced by random values, while the missing values of “body-style” and “drive-wheels” are replaced by “hatchback” and “fwd”, respectively. If the values of “length”, “width” or “height” are missing, the example is discarded. Values of all other attributes use the default method set above (model-based imputer, in our case).
3. The imputation methods for individual attributes are the same as default methods.
4. Restore All to Default resets the individual attribute treatments to default.
5. Produce a report.
6. All changes are committed immediately if Apply automatically is checked. Otherwise, Apply needs to be ticked to apply any new settings.

Example:

To demonstrate how the Impute widget works, we played around with the Iris dataset and deleted some of the data. We used the Impute widget and selected the Model-based imputer to impute the missing values. In another Data Table, we see how the question marks turned into distinct values (“Iris-setosa, “Iris-versicolor”).



7.8. Outliers

The outlier detection widget is used to classify the dataset.

7.8.1. Inputs

- Data: input dataset

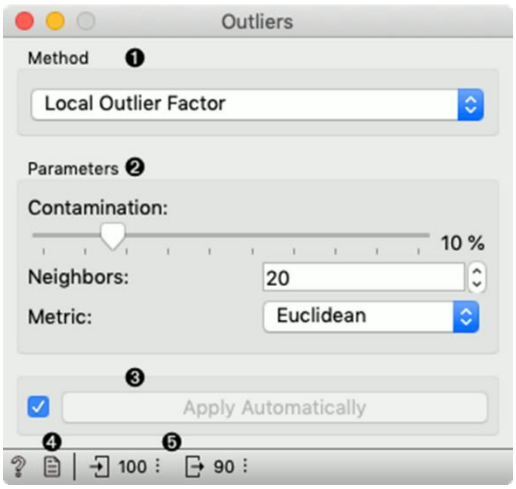
7.8.2. Outputs

- Outliers: instances scored as outliers
- Inliers: instances not scored as outliers
- Data: input dataset appended Outlier variable

The Outliers widget applies one of the four methods for outlier detection. All methods apply classification to the dataset. One-class SVM with non-linear kernels (RBF) performs well with non-Gaussian distributions, while the co-variance estimator works only for data with Gaussian distribution. One efficient way to perform outlier detection on moderately high dimensional datasets is to use the Local Outlier Factor algorithm. The algorithm computes a score reflecting the degree of abnormality of the observations. It measures the local density deviation of a given data point with respect to its neighbours. Another efficient way

of performing outlier detection in high-dimensional datasets is to use random forests (Isolation Forest).

Figure 7.12 Outliers



1. Method for outlier detection:
   - One Class SVM
   - Covariance Estimator
   - Local Outlier Factor
   - Isolation Forest
2. Set parameters for the method:
   - **One class SVM with the non-linear kernel (RBF):** classifies data as similar or different from the core class:
     – Nu is a parameter for the upper bound on the fraction of training errors and a lower bound of the fraction of support vectors
     – Kernel coefficient is a gamma parameter, which specifies how much influence a single data instance has
   - **Covariance estimator:** fits ellipsis to central points with Mahalanobis distance metric:
     – Contamination is the proportion of outliers in the dataset
     – Support fraction specifies the proportion of points included in the estimate
   - **Local Outlier Factor:** obtains local density from the k-nearest neighbours:
     – Contamination is the proportion of outliers in the dataset
     – Neighbours represent number of neighbours

  - Metric is the distance measure
- Isolation Forest: isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature:
  - Contamination is the proportion of outliers in the dataset
  - Replicable training fixes random seed

3. If Apply automatically is ticked, changes will be propagated automatically. Alternatively, click Apply.
4. Produce a report.
5. Number of instances on the input, followed by a number of instances scored as inliers.

Example:

Below is an example of how to use this widget. We used a subset (versicolor and virginica instances) of the Iris dataset to detect the outliers. We chose the Local Outlier Factor method, with Euclidean distance. Then we observed the annotated instances in the Scatter Plot widget. In the next step we used the setosa instances to demonstrate novelty detection using Apply Domain widget. After concatenating both outputs we examined the outliers in the Scatter Plot (1).

7.9. Pre-process

Pre-process data with selected methods and offers several preprocessing methods that can be combined in a single preprocessing pipeline.

7.9.1. Inputs

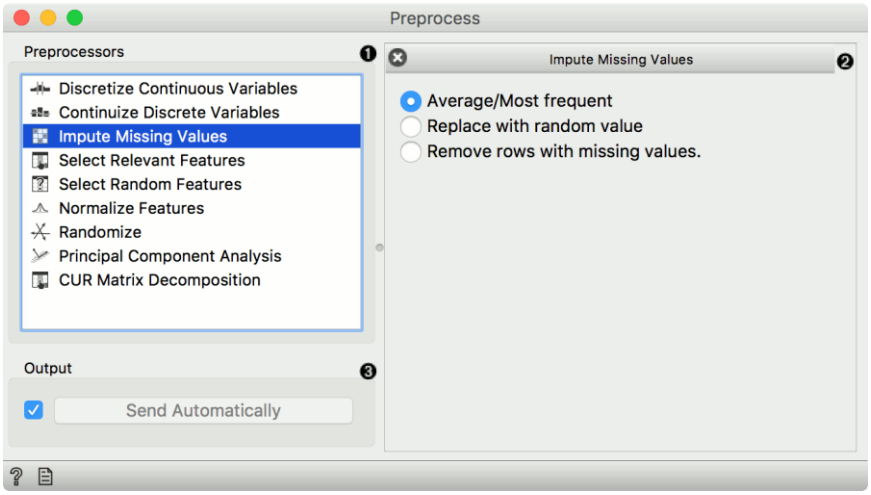
- Data: input dataset

7.9.2. Outputs

- Pre-processor: pre-processing method
- Pre-processed Data: data pre-processed with selected methods

Pre-processing is crucial for achieving better-quality analysis results. The Pre-process widget offers several pre-processing methods that can be combined into a single pre-processing pipeline. Some methods are available as separate widgets, which offer advanced techniques and greater parameter tuning.

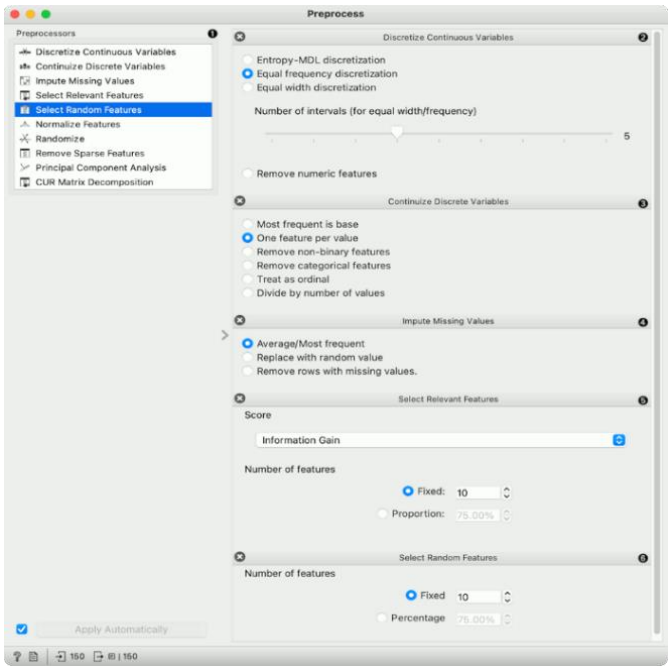
Figure 7.13 Impute Missing Values Preprocess



1. List of pre-processors. Double click the pre-processors you wish to use and shuÆe their order by dragging them up or down. You can also add pre-processors by dragging them from the left menu to the right.
2. Pre-processing pipeline.
3. When the box is ticked (Send Automatically), the widget will communicate changes automatically. Alternatively, click Send.

7.9.3. Pre-processors

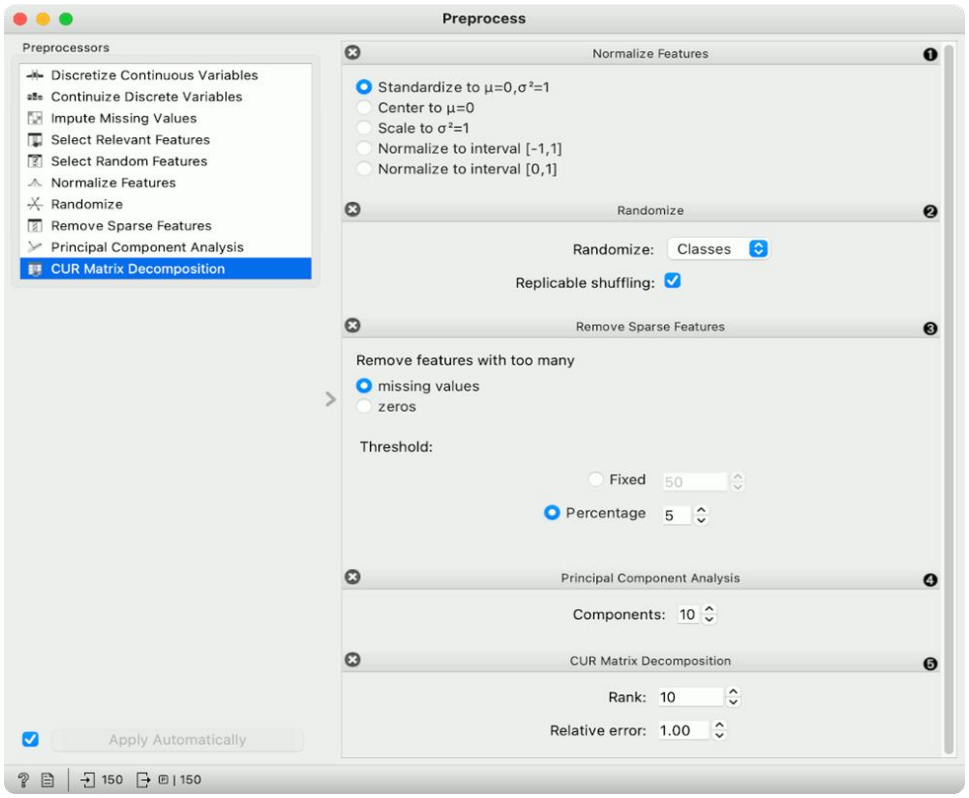
Figure 7.14 Select Random Features Preprocess



1. List of pre-processors.
2. Discretization of continuous values:
    - Entropy-MDL discretization by Fayyad and Irani that uses expected information to determine bins.
    - Equal frequency discretization splits by frequency (same number of instances in each bin.
    - Equal width discretization creates bins of equal width (span of each bin is the same).
    - Remove numeric features altogether.
3. Continuation of discrete values:
    - Most frequent as base treats the most frequent discrete value as 0 and others as 1. The discrete attributes with more than 2 values, the most frequent will be considered as a base and contrasted with the remaining values in corresponding columns.
    - One feature per value creates columns for each value, place 1 where an instance has that value and 0 where it doesn't. Essentially One Hot Encoding.
    - Remove non-binary features retain only categorical features that have values of either 0 or 1 and transform them into continuous.
    - Remove categorical features and removes categorical features altogether.
    - Treat as ordinal takes discrete values and treats them as numbers. If discrete values are categories, each category will be assigned a number as they appear in the data.
    - Divide by number of values is similar to treat as ordinal, but the final values will be divided by the total number of values and hence the range of the new continuous variable will be [0, 1].
4. Impute missing values:
    - Average/Most frequent replaces missing values (NaN) with the average (for continuous) or most frequent (for discrete) value.
    - Replace with random value replaces missing values with random ones within the range of each variable.
    - Remove rows with missing values.

5. Select relevant features:
    - Similar to Rank, this pre-processor outputs only the most informative features. A score can be determined by information gain, gain ratio, gini index, ReliefF, fast correlation-based filter, ANOVA, Chi2, RReliefF, and Univariate Linear Regression.
    - Strategy refers to how many variables should be on the output. Fixed returns a fixed number of top scored variables, while Percentile return the selected top percent of the features.
6. Select random features outputs either a fixed number of features from the original data or a percentage. This is mainly used for advanced testing and educational purposes.

Figure 7.15 CUR Matrix Decomposition



1. Normalize adjusts values to a common scale. Center values by mean or median or omit centering altogether. Similar for scaling, one can scale by SD (standard deviation), by span, or not at all.
2. Randomize instances. Randomized classes shuÆes class values and destroys the connection between instances and class. Similarly, one can randomize features or

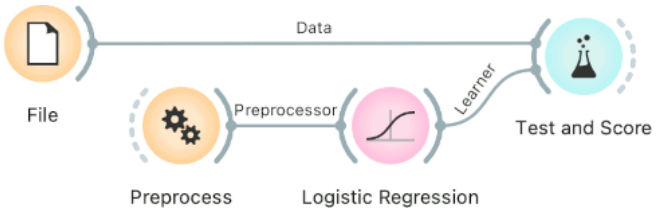
metadata. If replicable shuÆing is on, randomization results can be shared and repeated with a saved workflow. This is mainly used for advanced testing and educational purposes.

3. Remove sparse features retain features that have more than a number/percentage of non-zero/missing values. The rest are discarded.
4. Principal component analysis outputs results of a PCA transformation. Similar to the PCA widget.
5. CUR matrix decomposition is a dimensionality reduction method, similar to SVD.

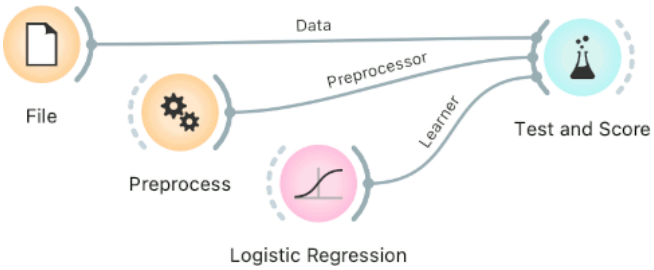
7.9.4. Pre-processing for predictive modelling

When building predictive models, one has to be careful about how to do pre-processing. There are two possible ways to do it in Orange, each slightly different:

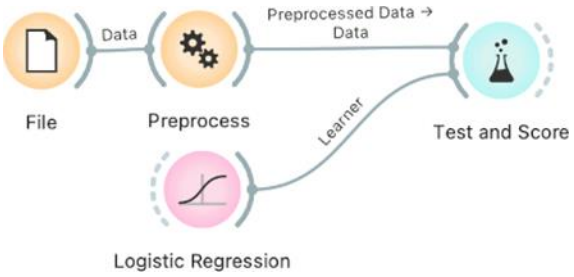
1. Connect Pre-process to the learner. This will override the default pre-processing pipeline for the learner and apply only custom pre-processing pipeline (default pre-processing steps are described in each learner's documentation).



2. Connect Pre-process to Test and Score. This will apply the pre-processors to each batch within cross-validation. Then the learner's pre-processors will be applied to the pre-processed subset.

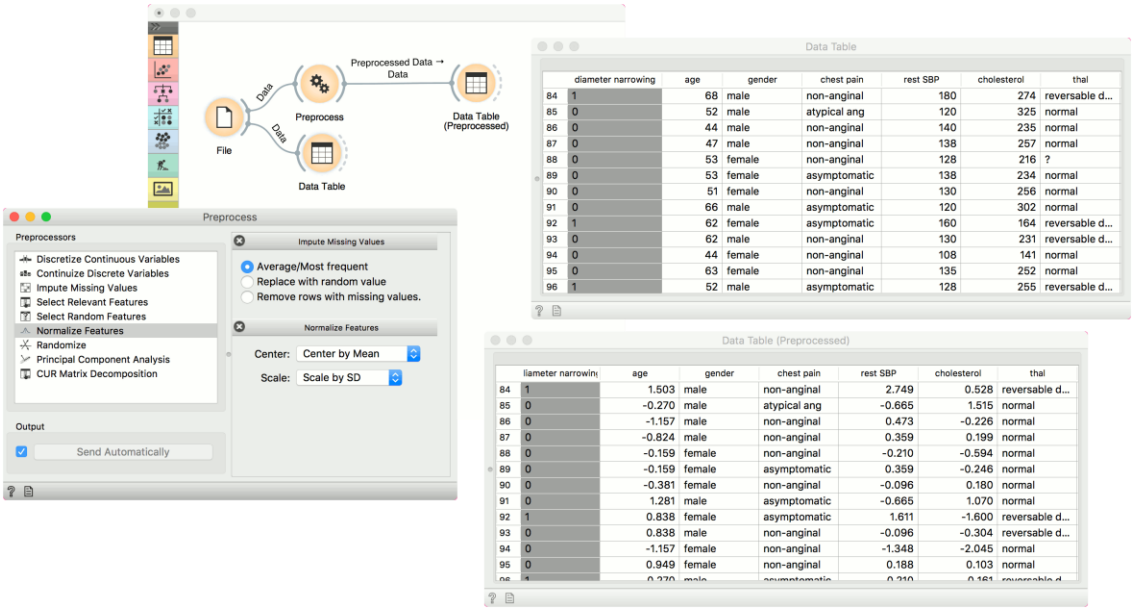


Finally, there's a wrong way to do it. Connecting Pre-process directly to the original data and outputting pre-processed data set will likely overfit the model. Don't do it.



Example:

In the first example, we have used the **heart_disease.tab** dataset available in the dropdown menu of the File widget. then we used Pre-process to impute missing values and normalize features. We can observe the changes in the Data Table and compare it to the non-processed data.



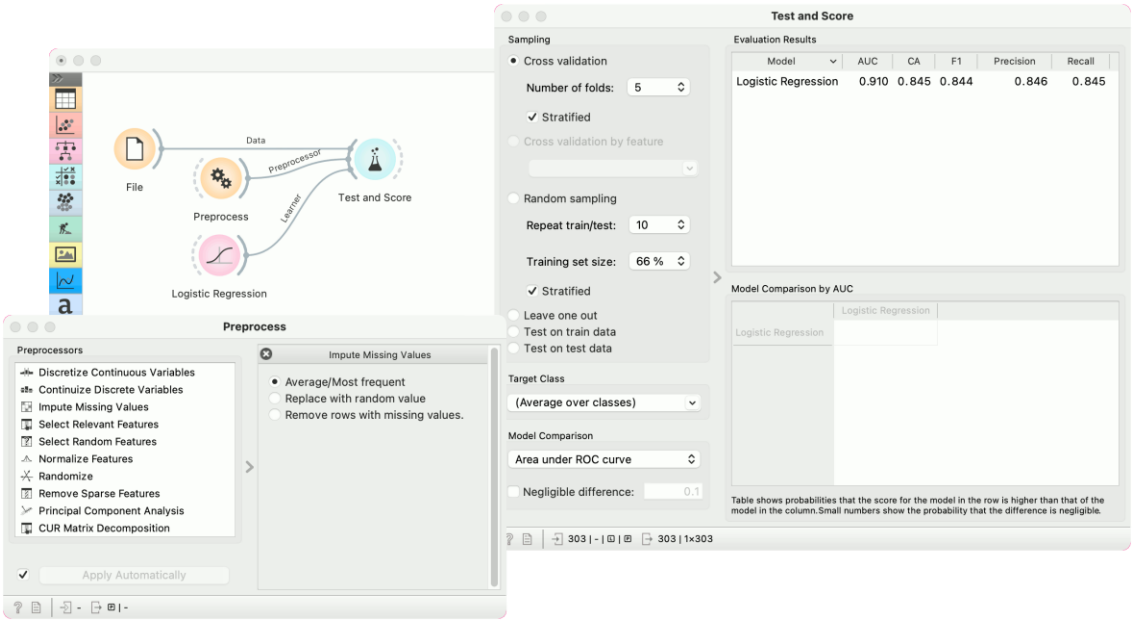
In the second example, we show how to use Preprocess for predictive modeling.

This time we are using the heart_disease.tab data from the File widget. You can access the data in the dropdown menu. This is a dataset with 303 patients that came to the doctor suffering from chest pain. After the tests were done, some patients were found to have diameter narrowing and others did not (this is our class variable).

Some values are missing in our data set, so we would like to impute missing values before evaluating the model. We do this by passing a preprocessor directly to Test and Score. In Preprocess, we set the correct preprocessing pipeline (in our example only a single

preprocessor with Impute missing values), then connect it to the Preprocessor input of Test and Score.

We also pass the data and the learner (in this case, a Logistic Regression). This is the correct way to pass a preprocessor to cross-validation as each fold will independently get preprocessed in the training phase. This is particularly important for feature selection.



Summary

1. Orange 3 is a tool that can analyze data simply without coding.
2. File and CSV File Import widgets can be used to retrieve and utilize files stored on the computer.
3. The Datasets widget can be used to retrieve and utilize various example data used for data analysis learning.
4. Data Table is a widget that converts data into a table form and shows it in detail.
5. Select Columns, Select Rows widgets allow you to select columns and rows according to specific conditions.
6. The Input widget can be used to fill in missing values (missing data) that may occur in the data.
7. The Outliers widget can be used to locate and exclude outliers (data outside of statistical scope).
8. Pre-process widget can be preprocessed in the desired form through various preprocessing functions such as data conversion and data modification.

Questions

1. What is the file extension in the form of storing data separated by commas?
2. What expression method has columns and rows and can accurately determine the exact value of the data?
3. What do rows and columns of data mean in the data expressed in table form?
4. Give 3 examples of how to fill in missing values of data.

Exercises

1. Use Google Links (bit.ly/o3knue) to upload ice cream sales data to Orange 3.
2. Express and interpret the ice cream sales data in table form.
3. Fill in the missing values from the ice cream sales data.

Chapter 08
Data visualization

In this chapter, you will learn about various ways to visualize your data. The larger the number of data, the more difficult it is to interpret it as a representation in the form of a table, and the more difficult it is to interpret the detailed interpretation of relationship between the data.

The advantage of representing data in charts, tables, and plots is that you can easily see and interpret data trends and features at a glance.

In this chapter, you will learn:

8.1. Box Plot

Shows the distribution of attribute values.

8.1.1. Inputs

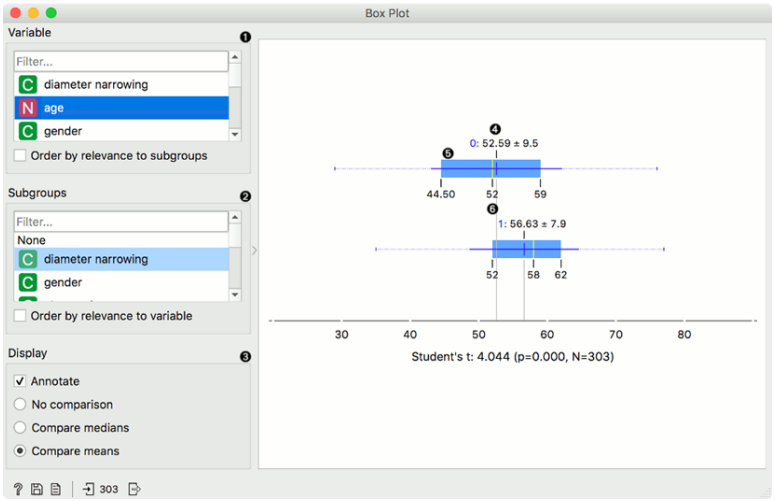
- Data: input dataset

8.2.2. Outputs

- Selected Data: instances selected from the plot
- Data: data with an additional column showing whether a point is selected

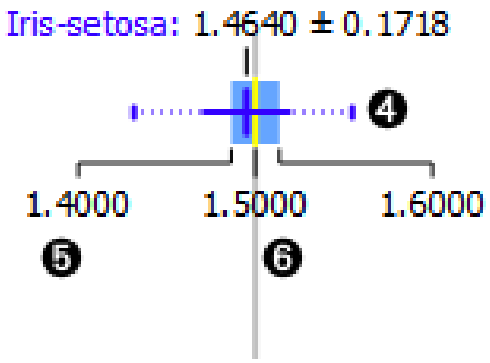
The Box Plot widget shows the distributions of attribute values. It is a good practice to check any new data with this widget to quickly discover any anomalies, such as duplicated values (e.g., gray and grey), outliers, and alike. Bars can be selected - for example, values for categorical data or the quantile range for numeric data.

Figure 8.1 Box Plot



1. Select the variable you want to plot. Tick Order by relevance to subgroups to order variables by Chi2 or ANOVA over the selected subgroup.
2. Choose Subgroups to see box plots displayed by a discrete subgroup. Tick Order by relevance to variable to order subgroups by Chi2 or ANOVA over the selected variable.
3. When instances are grouped by a subgroup, you can change the display mode. Annotated boxes will display the end values, the mean and the median, while comparing

medians and compare means will, naturally, compare the selected values between subgroups.



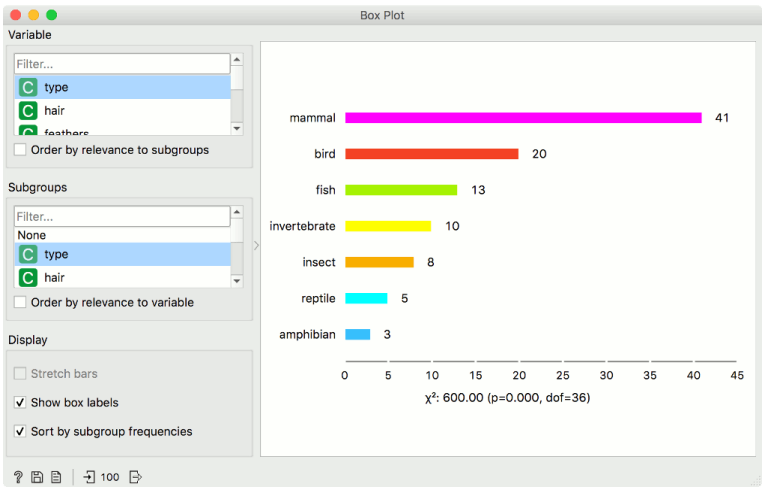
4. The mean (the dark blue vertical line). The thin blue line represents the standard deviation.
5. Values of the first (25%) and the third (75%) quantile. The blue highlighted area represents the values between the first and the third quartile.
6. The median (yellow vertical line).

For discrete attributes, the bars represent the number of instances with each particular attribute value. The plot shows the number of different animal types in the Zoo dataset: there are 41 mammals, 13 fish, 20 birds, and so on.

Display shows:

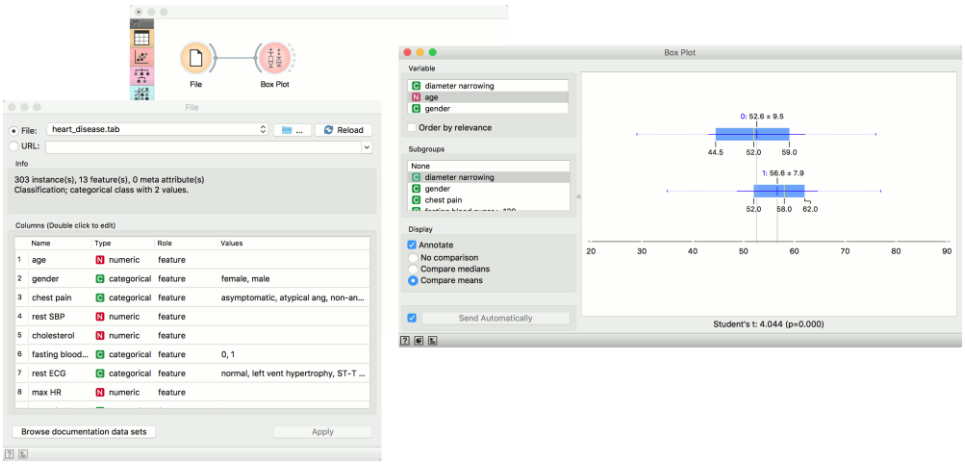
- Stretch bars: Shows relative values (proportions) of data instances. The unticked box shows absolute values.
- Show box labels: Display discrete values above each bar.
- Sort by subgroup frequencies: Sort subgroups by their descending frequency.

Figure 8.2 Variable



Examples:

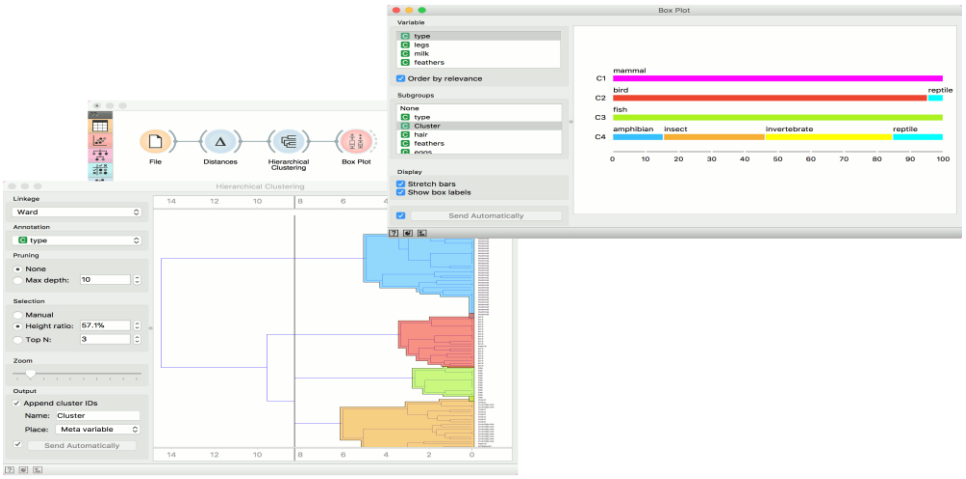
The Box Plot widget is most commonly used immediately after the File widget to observe the statistical properties of a dataset. In the first example, we used heart-disease data to inspect our variables.



Box Plot is also useful for finding the properties of a specific dataset, for instance, a set of instances manually defined in another widget (e.g. Scatter Plot or instances belonging to some cluster or a classification tree node. Let us now use zoo data and create a typical clustering workflow with Distances and Hierarchical Clustering.

Now define the threshold for cluster selection (click on the ruler at the top). Connect Box Plot to Hierarchical Clustering, tick Order by relevance, and select Cluster as a subgroup. This will order attributes by how well they define the selected subgroup, in our case, a cluster. It seems like our clusters indeed correspond very well with the animal type!

Figure 8.3 Hierarchical Clustering



8.2. Violin Plot

Visualize the distribution of feature values in a violin plot.

8.2.1. Inputs

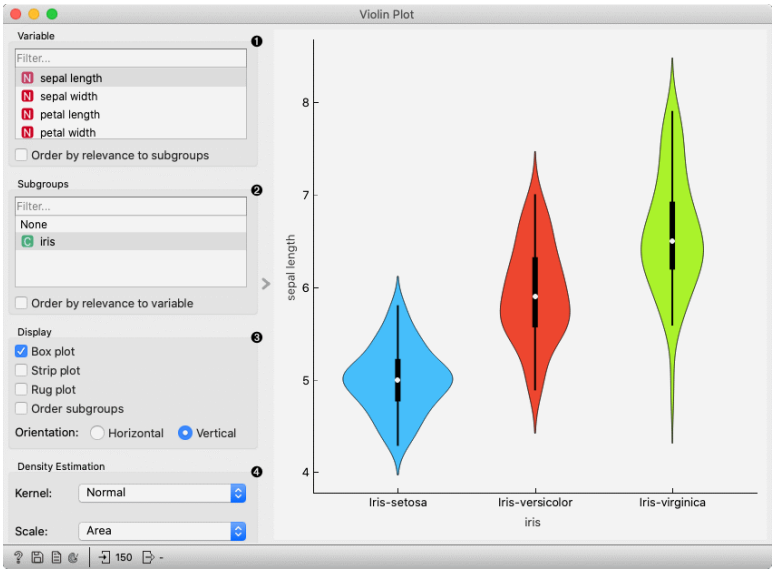
- Data: input dataset

8.2.2. Outputs

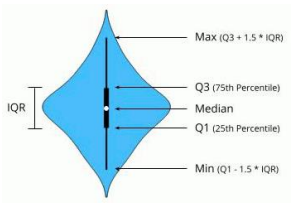
- Selected Data: instances selected from the plot
- Data: data with an additional column showing whether a point is selected

The Violin Plot widget plays a similar role as a Box Plot. It shows the distribution of quantitative data across several levels of a categorical variable such that those distributions can be compared. Unlike the Box Plot, in which all of the plot components correspond to actual data points, the Violin Plot features a kernel density estimation of the underlying distribution.

Figure 8.4 Violin Plot

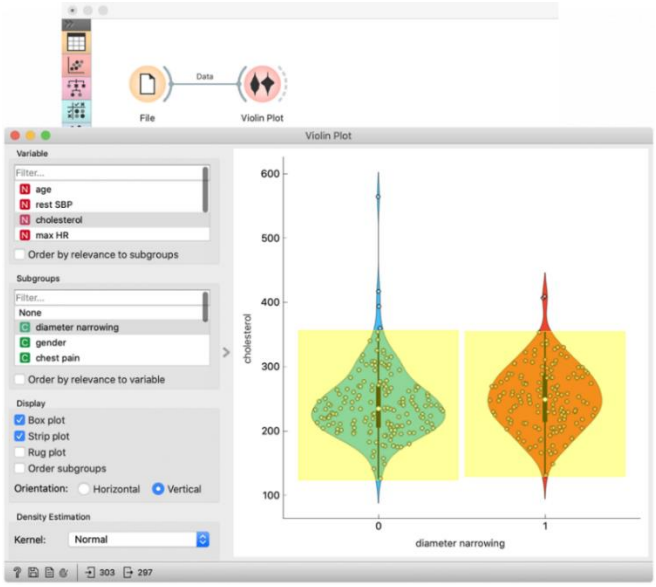


1. Select the variable you want to plot. Tick Order by relevance to subgroups to order variables by Chi2 or ANOVA over the selected subgroup.
2. Choose Subgroups to see violin plots displayed by a discrete subgroup. Tick Order by relevance to variable to order subgroups by Chi2 or ANOVA over the selected variable.
3. Box plot: Tick to show the underlying box plot.
   - Strip plot: Tick to show the underlying data represented by points.
   - Rug plot: Tick to show the underlying data represented by lines.



   - Order subgroups: Tick to order violins by median (ascending).
   - Orientation: Determine violin orientation.
4. Kernel: Select the kernel used to estimate the density. Possible kernels are: Normal, Epanechnikov and Linear.

Scale: Select the method used to scale the width of each violin. If area is selected, each violin will have the same area. If count is selected, the width of the violins will be scaled by the number of observations in that bin. If width is selected, each violin will have the same width.

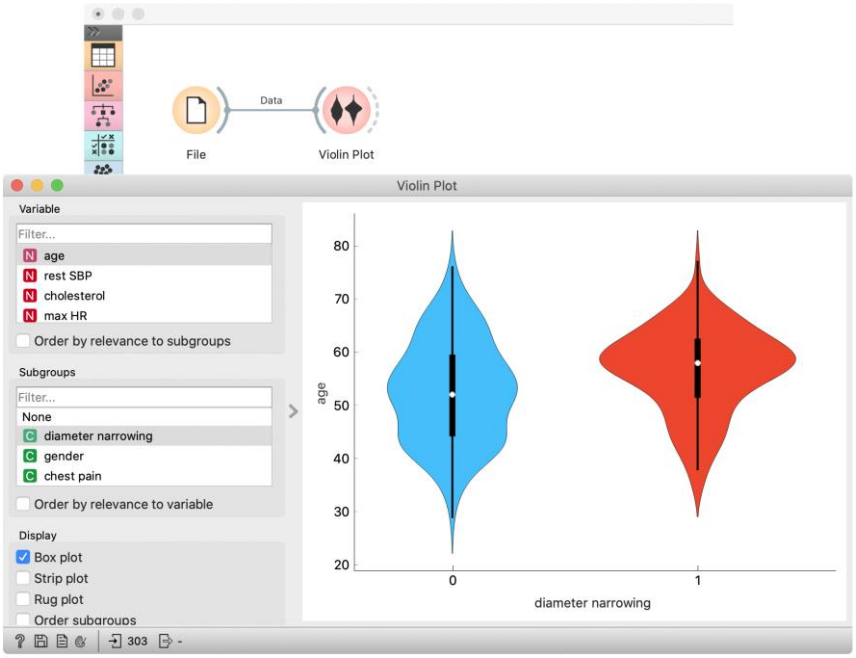


Example:

The Violin Plot widget is most commonly used immediately after the File widget to observe the statistical properties of a dataset. In the first example, we have used heart-disease data to inspect our variables.

The Violin Plot could also be used for outlier detection. In the next example, we eliminate the outliers by selecting only instances that fall inside the Q1 1.5 and Q3 + 1.5 IQR.

Figure 8.5 Violin Plot by Display Strip Plot



8.3. Distribution

Displays value distributions for a single attribute.

8.3.1. Inputs

- Data: input dataset

8.3.1. Outputs

– Selected Data: instances selected from the plot

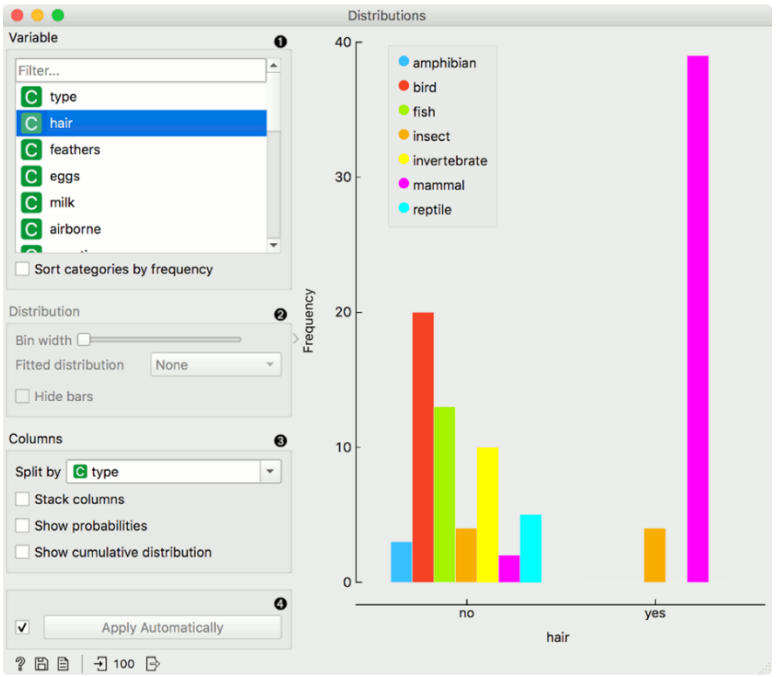
– Data: data with an additional column showing whether an instance is selected

– Histogram Data: bins and instance counts from the histogram

The Distributions widget displays the value distribution of discrete or continuous attributes. If the data contains a class variable, distributions may be conditioned on the class.

The graph shows how many times (e.g., in how many instances) each attribute value appears in the data. If the data contains a class variable, class distributions for each of the attribute values will be displayed (like in the snapshot below). To create this graph, we used the Zoo dataset.

Figure 8.6 Distributions



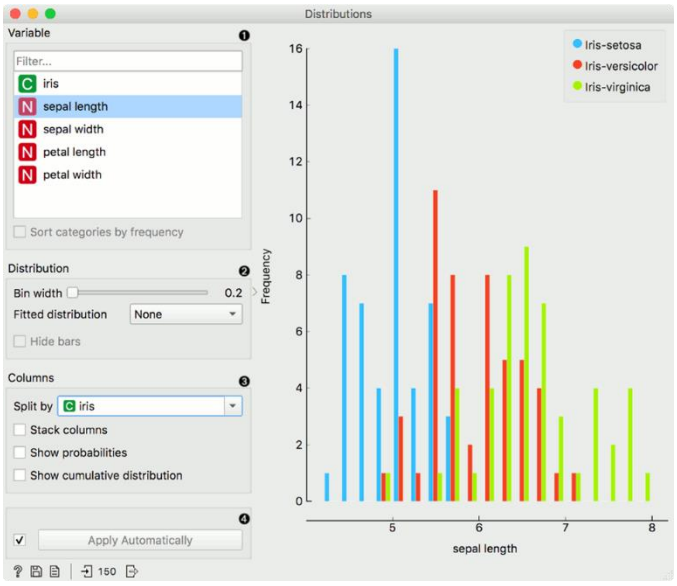
1. A list of variables for display. Sort categories by frequency orders displayed values by frequency.
2. Set Bin width with the slider. Precision scale is set to sensible intervals. Fitted distribution fits the selected distribution to the plot. Options are Normal, Beta, Gamma, Rayleigh, Pareto, Exponential, Kernel density.
3. Columns:
    - Split by displays value distributions for instances of a certain class.
    - Stack columns displays one column per bin, colored by proportions of class values.
    - Show probabilities shows probabilities of class values at selected variable.
    - Show cumulative distribution cumulatively stacks frequencies.

If Apply Automatically is ticked, changes are communicated automatically. Alternatively, click Apply.

For continuous attributes, the attribute values are also displayed as a histogram. It is possible to fit various distributions to the data, for example, a Gaussian kernel density estimation. Hide bars hides histogram bars and shows only distribution (old behaviour of Distributions).

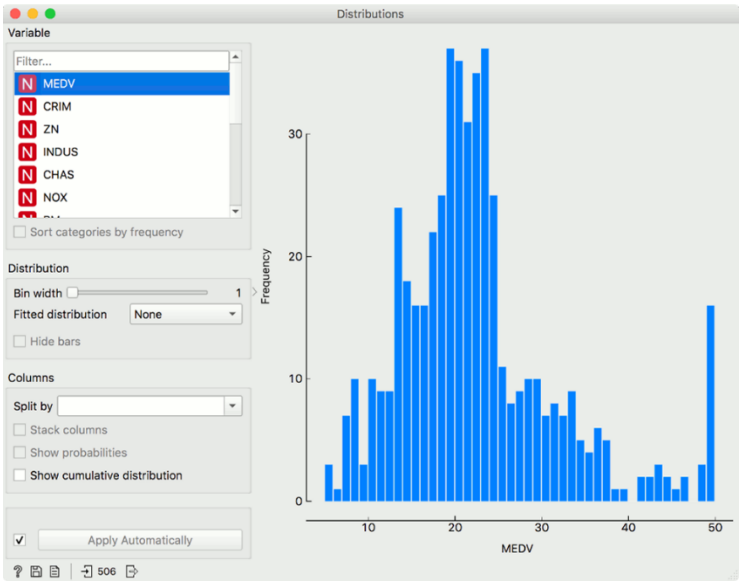
For this example, we used the Iris dataset.

Figure 8.7 Distributions by Sepal Length



In classless domains, the bars are displayed in blue. We used the Housing dataset.

Figure 8.8 Distributions by MEDV



8.4. Heat Map

Plots a heat map for a pair of attributes and is a graphical method for visualizing attribute values in a two-way matrix.

8.4.1 Inputs

- Data: input dataset

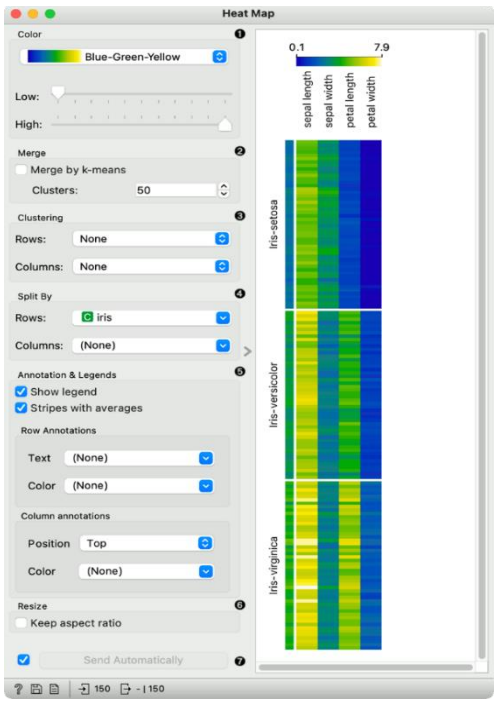
8.4.2. Outputs

- Selected Data: instances selected from the plot
- Data: data with an additional column showing whether a point is selected

Heat map is a graphical method for visualizing attribute values in a two-way matrix. It only works on datasets containing numeric variables. The values are represented by color according to the selected color palette. By combining class variables and attributes on the x and y axes, we see where the attribute values are the strongest and where the weakest, thus enabling us to find typical features for each class.

The widget enables row selection with click and drag. One can zoom in with Ctrl++ (Cmd++) and zoom out with Ctrl+- (Cmd+-). Ctrl+0 (Cmd+0) resets zoom to the extended version, while Ctrl+9 (Cmd+9) reset it to the default.

Figure 8.9 Heat Map



1. The color palette. Choose from linear, diverging, color-blind-friendly, or other palettes. Low and High are thresholds for the color palette (low for attributes with low values and high for attributes with high values). Selecting one of the diverging palettes, which have two extreme colors and a neutral (black or white) color at the midpoint, enables an option to set a meaningful mid-point value (default is 0).
2. Merge rows. If there are too many rows in the visualization, one can merge them with k-means algorithm into N-selected clusters (default 50).
3. Cluster columns and rows:
    - None (lists attributes and rows as found in the dataset)
    - Clustering (clusters data by similarity with hierarchical clustering on Euclidean distances and with average linkage)
    - Clustering with ordered leaves (same as clustering, but it additionally maximizes the sum of similarities of adjacent elements)
4. Split rows or columns by a categorical variable. If the data contains a class variable, rows will be automatically split by class.
5. Set what is displayed in the plot in Annotation & Legend.
    - If the show legend is ticked, a color chart will be displayed above the map.

- If Stripes with averages is ticked, a new line with attribute averages will be displayed on the left. Row An- notations add annotations to each instance on the right. Color colors the instances with the corresponding value of the selected categorical variable. Column Annotations adds an annotation to each variable at the selected position (default is Top). Color colors the columns with the corresponding value of the selected column annotation.

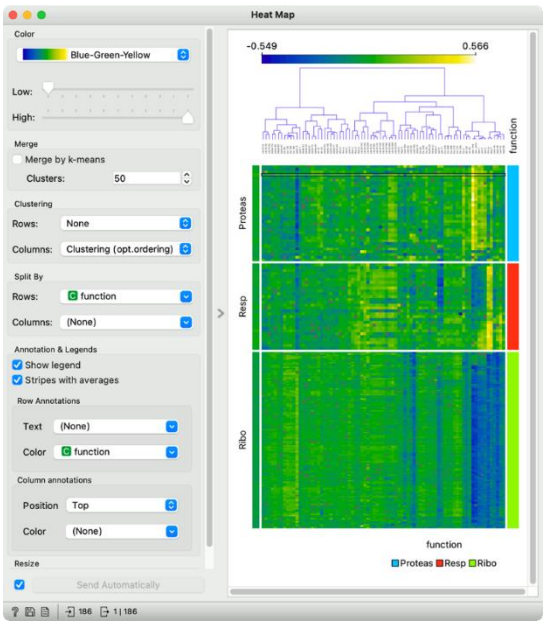
6. The If the keep aspect ratio is ticked, each value will be displayed with a square (proportionate to the map).
7. If Send Automatically is ticked, changes are communicated automatically. Alternatively, click Send.

8.4.3 Advanced visualization

Heat map enables some neat plot enhancements. Such options are clustering of rows and/or columns for better data organization, row, and column annotations, and splitting the data by categorical variables.

Row and column clustering is performed independently. Row clustering is computed from Euclidean distances, while column clustering uses Pearson correlation coefficients. Hierarchical clustering is based on the Ward linkage method. Clustering with optimal leaf ordering reorders left and right branches in the dendrogram to minimize the sum of distances between adjacent leaves (Bar-Joseph et al. 2001).

Figure 8.10 Advanced Heat Map



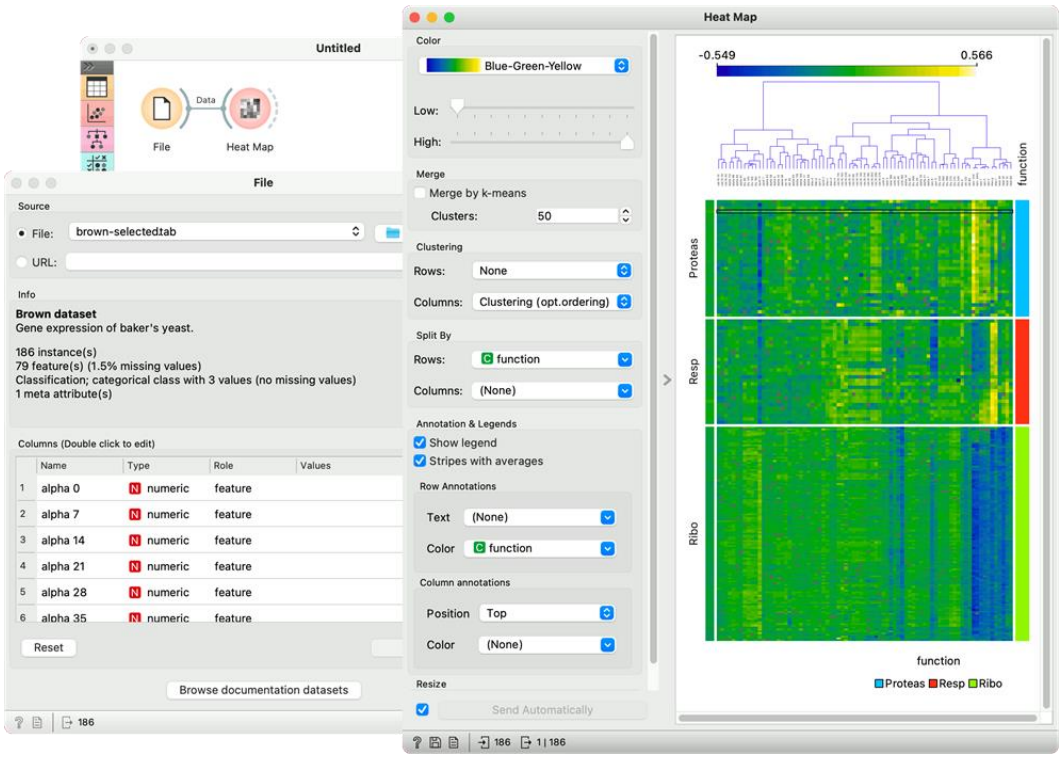
8.4.4. Gene expressions

The Heat Map below displays attribute values for the brown-selected data set (Brown et al. 2000). Heat maps are particularly appropriate for showing gene expressions and the brown-selected data set contains yeast gene expressions at different conditions.

Heat map shows low expressions in blue and high expressions in yellow and white. For better organization, we added Clustering (opt. ordering) to the columns, which puts columns with similar profiles closer together. In this way, we can see the conditions that result in low expressions for ribosomal genes in the lower right corner.

Additionally, the plot is enhanced with row color on the right, showing which class the rows belong to.

Figure 8.11 Advanced Heat Map by Gene expression



8.4.5. Sentiment Analysis

Heat maps are great for visualizing any kind of comparable numeric variables, for example, sentiment in a collection of documents. We will take the book-excerpts corpus from the Corpus widget and pass it to the Sentiment Analysis widget, which computes sentiment scores for each document. The output of sentiment analysis is four columns,

positive, negative, and neutral sentiment score, and a compound score that aggregates the previous scores into a single number. Positive compound values (white) represent positive documents, while a negative (blue) represent negative documents.

We used row clustering to place similar rows closer together, resulting in clear negative and positive groups. Now we can select negative children's books and explore which are they.

Figure 8.12 Advanced Heat Map by Sentiment Analysis

8.4.6. References

Bar-Joseph, Z., Gifford, D.K., Jaakkola, T.S. (2001) Fast optimal leaf ordering for hierarchical clustering, Bioinformat- ics, 17, 22-29.

Brown, M.P., Grundy, W.N., Lin, D., Cristianini, N., Sugnet, C., Furey, T.S., Ares, M., Haussler, D. (2000) Knowledge- based analysis of microarray gene expression data by using support vector machines, Proceedings of the National Academy of Sciences, 1, 262-267.

8.5. Scatter Plot

Scatter plot visualization with exploratory analysis and intelligent data visualization enhancements.

8.5.1 Inputs

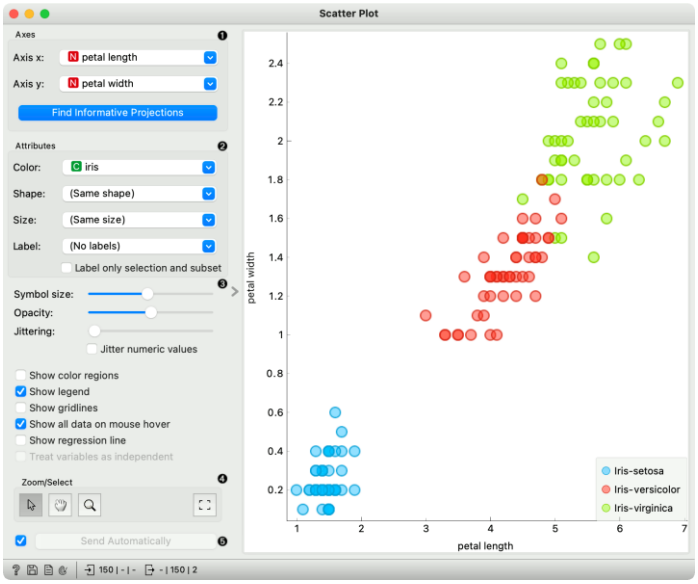
- Data: input dataset
- Data Subset: subset of instances
- Features: list of attributes

8.5.2. Outputs

- Selected Data: instances selected from the plot

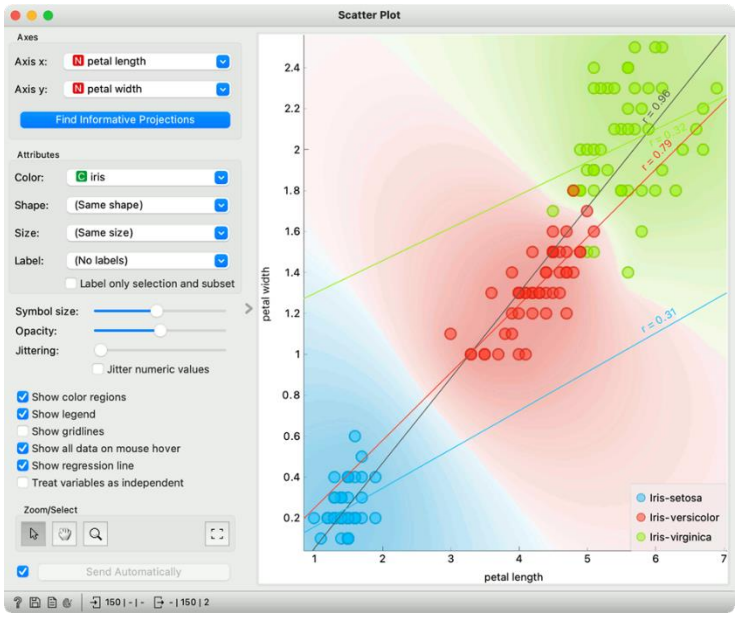
The Scatter Plot widget provides a 2-dimensional scatter plot visualization. The data is displayed as a collection of points, each having the value of the x-axis attribute determining the position on the horizontal axis and the value of the y-axis attribute determining the position on the vertical axis. Various properties of the graph, like color, size and shape of the points, axis titles, maximum point size, and jittering can be adjusted on the left side of the widget. A snapshot below shows the scatter plot of the Iris dataset with the coloring matching of the class attribute.

Figure 8.13 Scatter Plot



1. Select the x and y attribute. Optimize your projection with Find Informative Projections. This feature scores attribute pairs by average classification accuracy and returns the top-scoring pair with a simultaneous visualization update.
2. Attributes: Set the color of the displayed points (you will get colors for categorical values and blue-green-yellow points for numeric). Set label, shape, and size to differentiate between points. Label-only selected points to allow you to select individual data instances and label only those.
3. Set symbol size and opacity for all data points. Set jittering to prevent the dots from overlapping. Jittering will randomly scatter points only around categorical values. If Jitter numeric values are checked, points are also scattered around their actual numeric values.
    - Show color regions colors the graph by class (see the screenshot below).
    - Show legend displays a legend on the right. Click and drag the legend to move it.
    - Show gridlines display the grid behind the plot.
    - Show all data on mouse hover enabling information bubbles if the cursor is placed on a dot.
    - Show regression line draws the regression line for pair of numeric attributes. If a categorical variable is selected for coloring the plot, individual regression lines for each class value will be displayed. The reported r value corresponds to the value from linear least-squares regression, which is equal to Pearson's correlation coefficient.
    - Treat variables as independent fits regression line to a group of points (minimize the distance from points), rather than fitting y as a function of x (minimize vertical distances)
4. Select, zoom, pan, and zoom to fit are the options for exploring the graph. The manual selection of data instances works as an angular/square selection tool. Double-click to move the projection. Scroll in or out for zoom.
5. If Send automatically is ticked, changes are communicated automatically. Alternatively, press Send.

Here is an example of the Scatter Plot widget if the Show color regions and Show regression line boxes are ticked.



8.5.3. Intelligent Data Visualization

If a dataset has many attributes, it is impossible to manually scan through all the pairs to find interesting or useful scatter plots. Orange implements intelligent data visualization with the Find Informative Projections option in the widget.

If a categorical variable is selected in the Color section, the score is computed as follows. For each data instance, the method finds 10 nearest neighbors in the projected 2D space, that is, on the combination of attribute pairs. It then checks how many of them have the same color. The total score of the projection is then the average number of same-colored neighbors.

The computation for numeric colors is similar, except that the coefficient of determination is used for measuring the local homogeneity of the projection.

To use this method, go to the Find Informative Projections option in the widget, open the sub-window, and press Start Evaluation. The feature will return a list of attribute pairs by average classification accuracy score.

Below, there is an example demonstrating the utility of ranking. The first scatter plot projection was set as the default sepal width to sepal length plot (we used the Iris dataset for simplicity). Upon running Find Informative Projections optimization, the scatter plot converted to a much better projection of petal width to petal length plot.

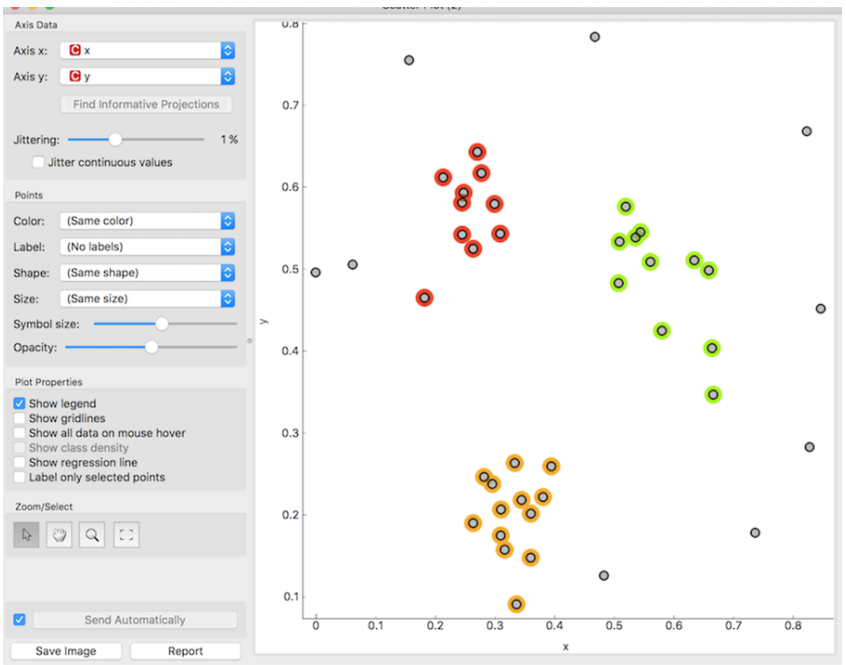
Figure 8.14 Intelligent Data Visualization

8.5.4. Selection

Selection can be used to manually defined subgroups in the data. Use Shift modifier when selecting data instances to put them into a new group. Shift + Ctrl (or Shift + Cmd on macOs) appends instances to the last group.

Signal data outputs a data table with an additional column that contains group indices.

Figure 8.15 Selection

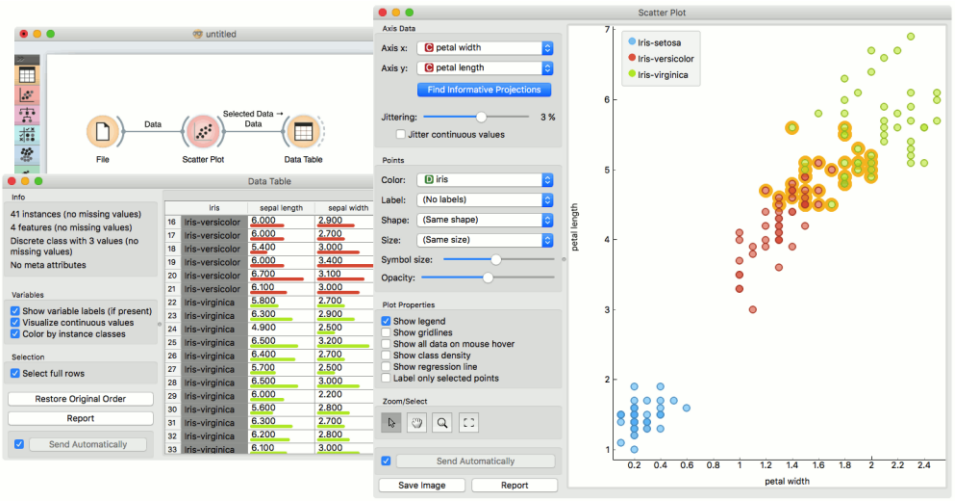


8.5.6. Exploratory Data Analysis

The Scatter Plot, like the rest of the Orange widgets, supports zooming in and out of part of the plot and a manual selection of data instances. These functions are available in the lower-left corner of the widget.

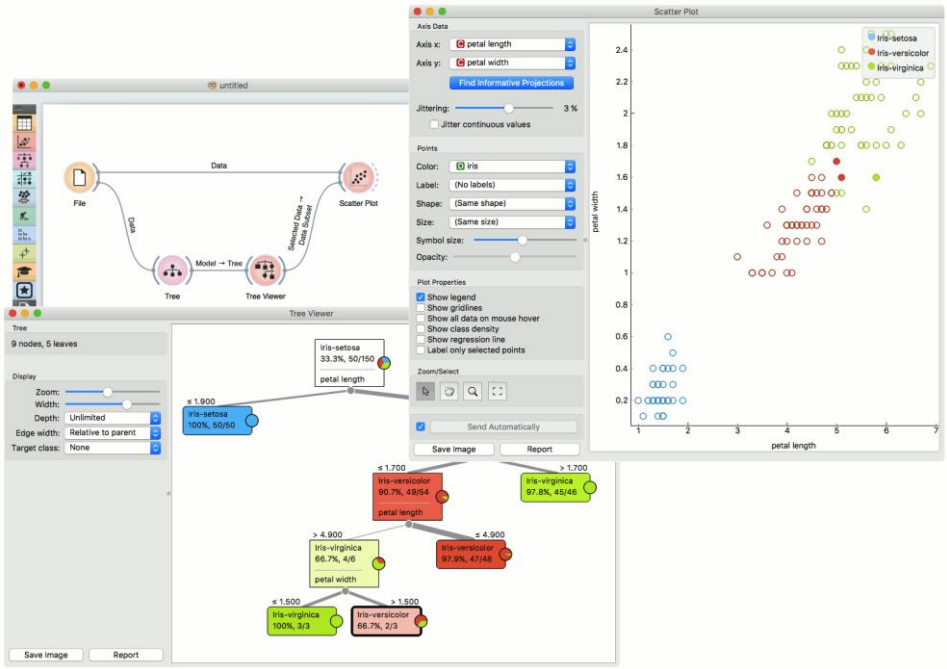
The default tool is Select, which selects data instances within the chosen rectangular area. Pan enables you to move the scatter plot around the pane. With Zoom, you can zoom in and out of the pane with a mouse scroll, while Reset zoom resets the visualization to its optimal size. An example of a simple schema, where we selected data instances from a rectangular region and sent them to the Data Table widget, is shown below. Notice that the scatter plot doesn't show all 52 data instances, because some data instances overlap (they have the same values for both attributes used).

Figure 8.16 Exploratory Data Analysis



Example:

The Scatter Plot can be combined with any widget that outputs a list of selected data instances. In the example below, we combine Tree and Scatter Plot to display instances taken from a chosen decision tree node (clicking on any node of the tree will send a set of selected data instances to the scatter plot and mark selected instances with filled symbols).



8.5.7. References

Gregor Leban and Blaz Zupan and Gaj Vidmar and Ivan Bratko (2006) VizRank: Data Visualization Guided by Ma- chine Learning. Data Mining and Knowledge Discovery, 13 (2). pp. 119-136. Available here.

8.6. Line Plot

It is a visualization of data profiles. A line Plot is a type of plot which displays the data as a series of points, connected by straight line segments.

8.6.1. Inputs

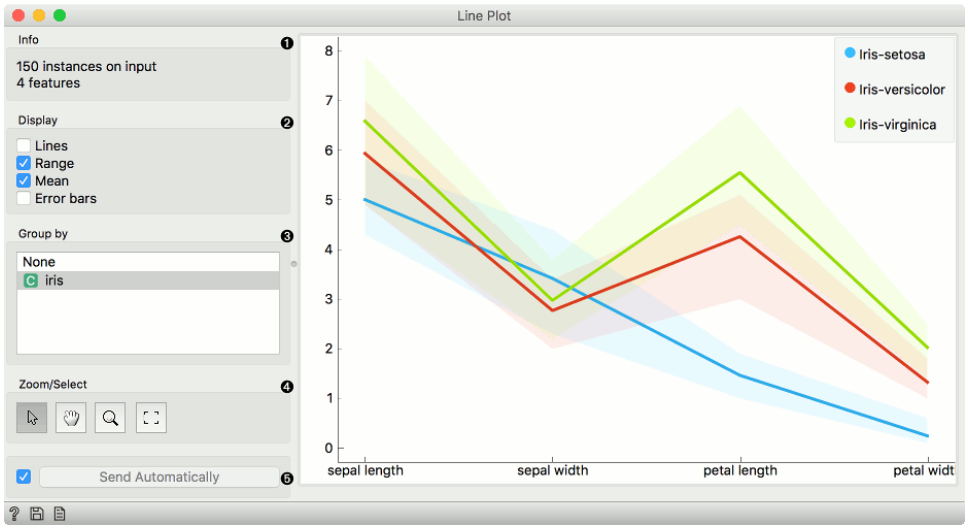
- Data: input dataset
- Data Subset: the subset of instances

8.6.1. Outputs

- Selected Data: instances selected from the plot
- Data: data with an additional column showing whether a point is selected

Line plot is a type of plot which displays the data as a series of points, connected by straight line segments. It only works for numerical data, while categorical can be used for grouping of the data points.

Figure 8.17 Line Plot



1. Information on the input data.
2. Select what you wish to display:
   - Lines show individual data instances in a plot.
   - Range shows the range of data points between the 10th and 90th percentile.
   - Mean adds the line for mean value. If a group by is selected, means will be displayed per each group value.
   - Error bars show the standard deviation of each attribute.
3. Select a categorical attribute to use for grouping data instances. Use None to show ungrouped data.
4. Select, zoom, pan, and zoom to fit are the options for exploring the graph. The manual selection of data instances works as a line selection, meaning the data under the selected line plots will be sent on the output. Scroll in or out for zoom. When hovering over an individual axis, scrolling will zoom only by the hovered-on axis (vertical or horizontal zoom).
5. If Send Automatically is ticked, changes are communicated automatically. Alternatively, click Send.

Example:

Line Plot is a standard visualization widget, which displays data profiles, normally of ordered numerical data. In this simple example, we will display the iris data in a line plot, grouped by the iris attribute. The plot shows how petal length nicely separates between class values.

If we observe this in a Scatter Plot, we can confirm this is indeed so. Petal length is an interesting attribute for separation of classes, especially when enhanced with petal width, which is also nicely separated in the line plot.

8.7. Bar Plot

Visualizes comparisons among discrete categories.

8.7.1. Inputs

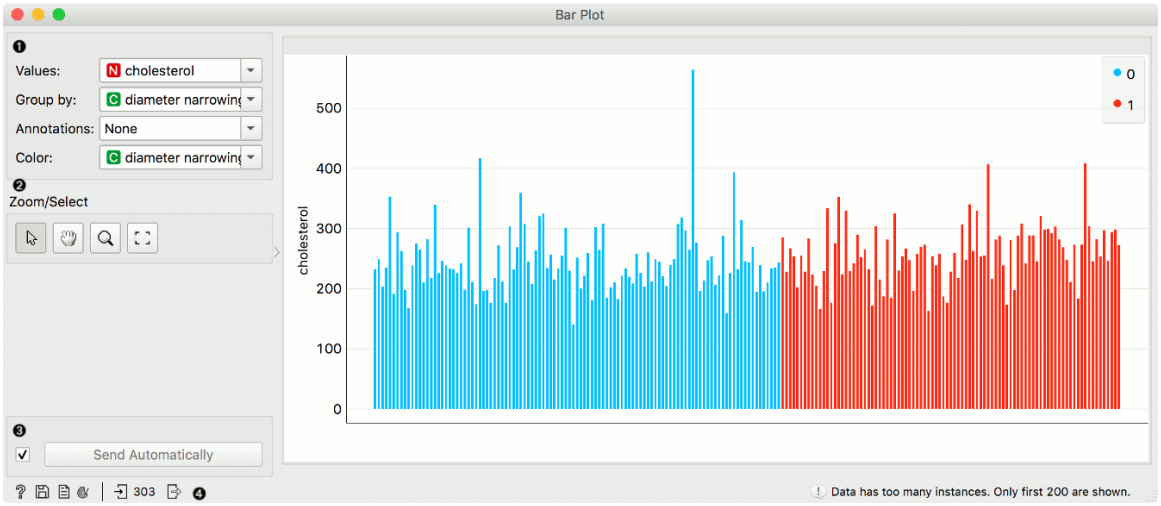
- Data: input dataset
- Data Subset: the subset of instances

8.7.2. Outputs

- Selected Data: instances selected from the plot
- Data: data with an additional column showing whether a point is selected

The Bar Plot widget visualizes numeric variables and compares them by a categorical variable. The widget is useful for observing outliers, distributions within groups, and comparing categories.

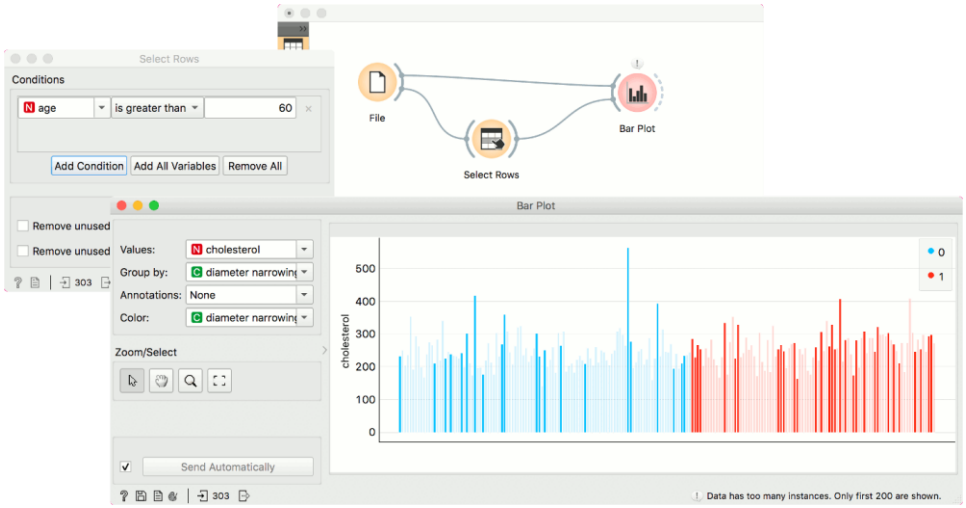
Figure 8.18 Bar Plot



1. Parameters of the plot. Values are the numeric variable to plot. Group by is the variable for grouping the data. Annotations are categorical labels below the plot. Color is the categorical variable whose values are used for colouring the bars.
2. Select, zoom, pan and zoom to fit are the options for exploring the graph. The manual selection of data instances works as an angular/square selection tool. Double click to move the projection. Scroll in or out for zoom.
3. If Send automatically is ticked, changes are communicated automatically. Alternatively, press Send.
4. Access help, save image, produce a report, or adjust visual settings. On the right, the information on input and output are shown.

Example:

The Bar Plot widget is most commonly used immediately after the File widget to compare categorical values. In this example, we have used heart-disease data to inspect our variables.



First, we observed the cholesterol values of patients from our data set. We grouped them by diameter narrowing, which defines patients with heart disease (1) and those without (0). We use the same variable for colouring the bars.

Then, we selected patients over 60 years of age with Select Rows. We sent the subset to Bar Plot to highlight these patients in the widget. The big outlier with a high cholesterol level is apparently over 60 years old.

8.8. Venn Diagram

Plots a Venn diagram for two or more data subsets.

8.8.1. Input

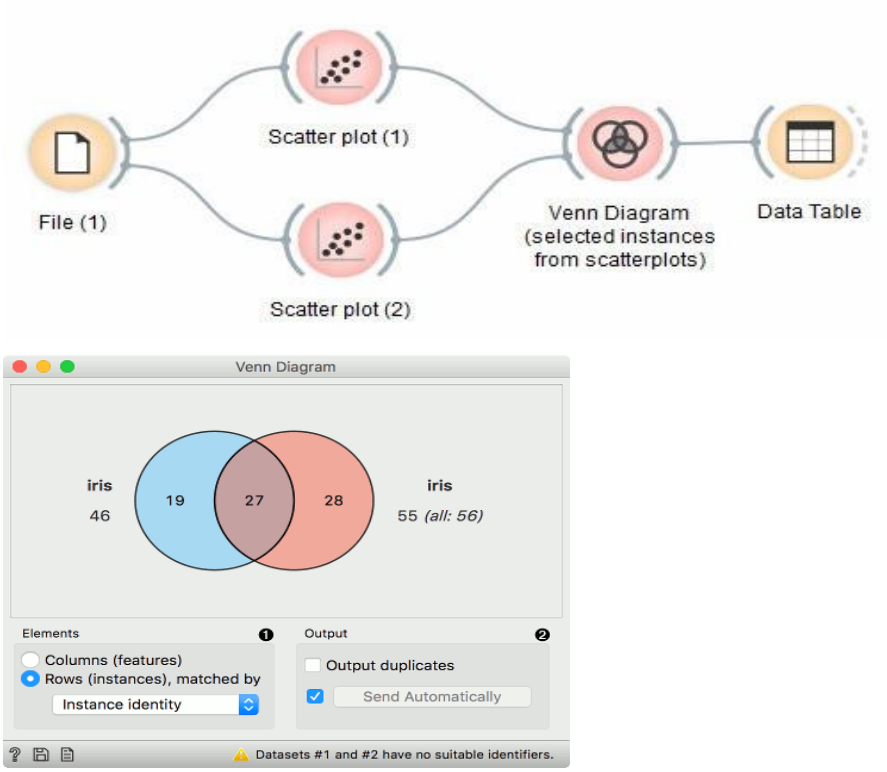
- Data: input dataset

8.8.2. Output

- Selected Data: instances selected from the plot
- Data: entire data with a column indicating whether an instance was selected or not

The Venn Diagram widget displays logical relations between datasets by showing the number of common data in- stances (rows) or the number of shared features (columns). Selecting a part of the visualization outputs the corresponding instances or features.

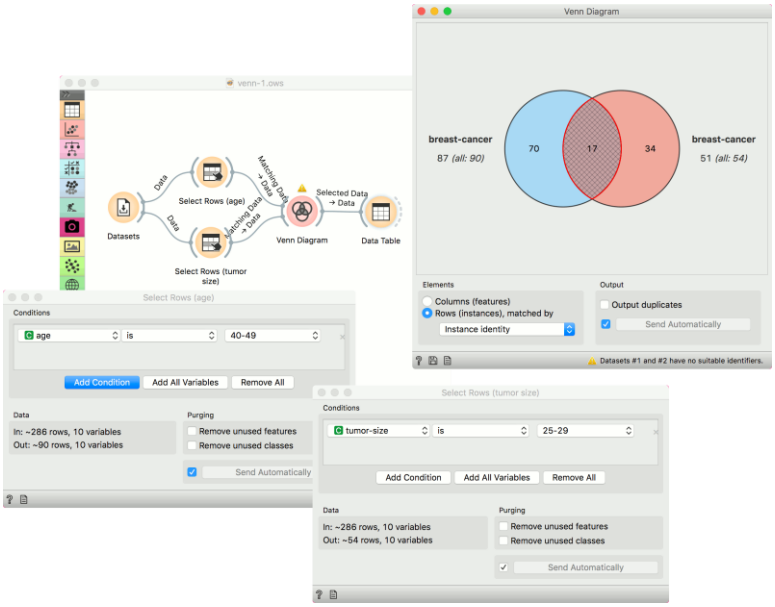
Figure 8.19 Venn Plot



1. Select whether to count common features or instances.
2. Select whether to include duplicates or to output only unique rows; applicable only when matching instances by values of variables.
3. Rows can be matched
   - by their identity, e.g. rows from different data sets match if they came from the same row in a file,
   - by equality, if all tables contain the same variables,
   - or by values of a string variable that appears in all tables.

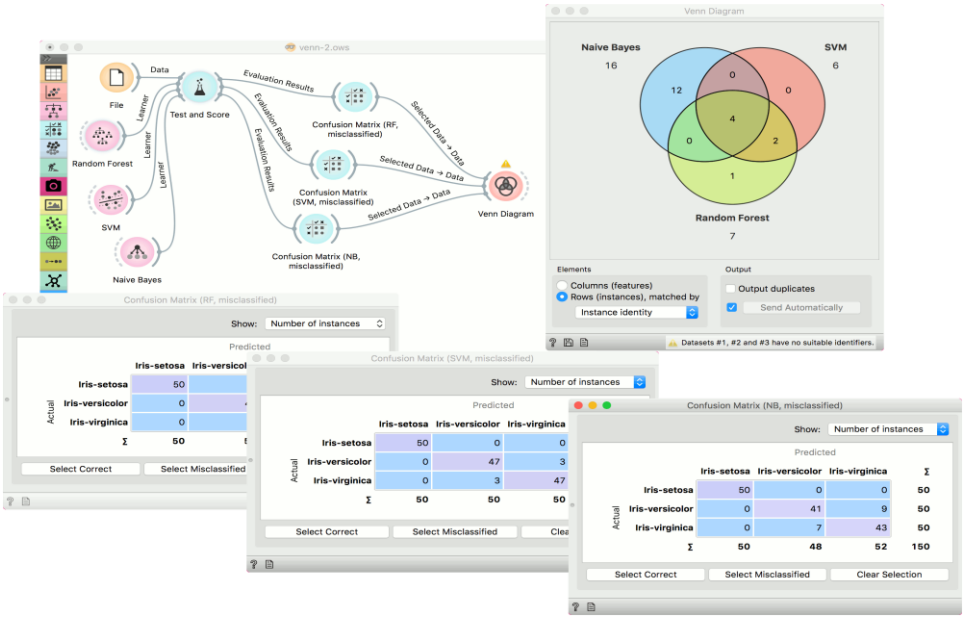
Example:

The easiest way to use the Venn Diagram is to select data subsets and find matching instances in the visualization. We use the breast-cancer dataset to select two subsets with Select Rows widget - the first subset is that of breast cancer patients aged between 40 and 49 and the second is that of patients with a tumor-size between 20 and 29. The Venn Diagram helps us find instances that correspond to both criteria, which can be found in the intersection of the two circles.



The Venn Diagram widget can be also used for exploring different prediction models. In the following example, we analysed 3 prediction methods, namely Naive Bayes, SVM and Random Forest, according to their misclassified instances.

By selecting misclassifications in the three Confusion Matrix widgets and sending them to Venn diagram, we can see all the misclassification instances visualized per method used. Then we open Venn Diagram and select, for example, the misclassified instances that were identified by all three methods. This is represented as an intersection of all three circles. Click on the intersection to see these two instances marked in the Scatter Plot widget. Try selecting different diagram sections to see how the scatter plot visualization changes.



8.9. Linear Projection

A linear projection method with explorative data analysis.

8.9.1. Inputs

- Data: input dataset
- Data Subset: subset of instances
- Projection: custom projection vectors

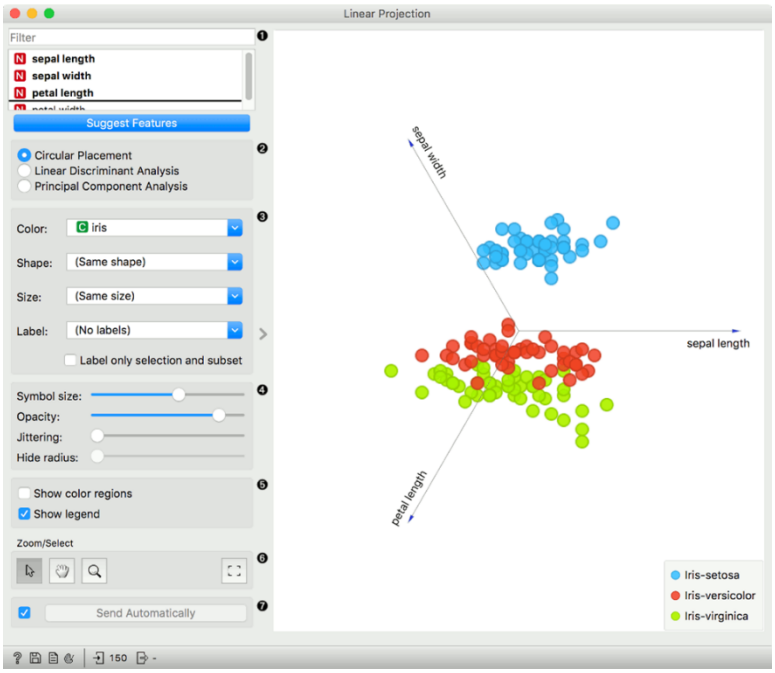
8.9.2. Outputs

- Selected Data: instances selected from the plot
- Data: data with an additional column showing whether a point is selected
- Components: projection vectors

This widget displays linear projections of class-labeled data. It supports various types of projections such as circular, linear discriminant analysis, and principal component analysis.

Consider, for a start, a projection of the Iris dataset shown below. Notice that it is the sepal width and sepal length that already separate Iris setosa from the other two, while the petal length is the attribute best separating Iris versicolor from Iris virginica.

Figure 8.20 Linear Projection

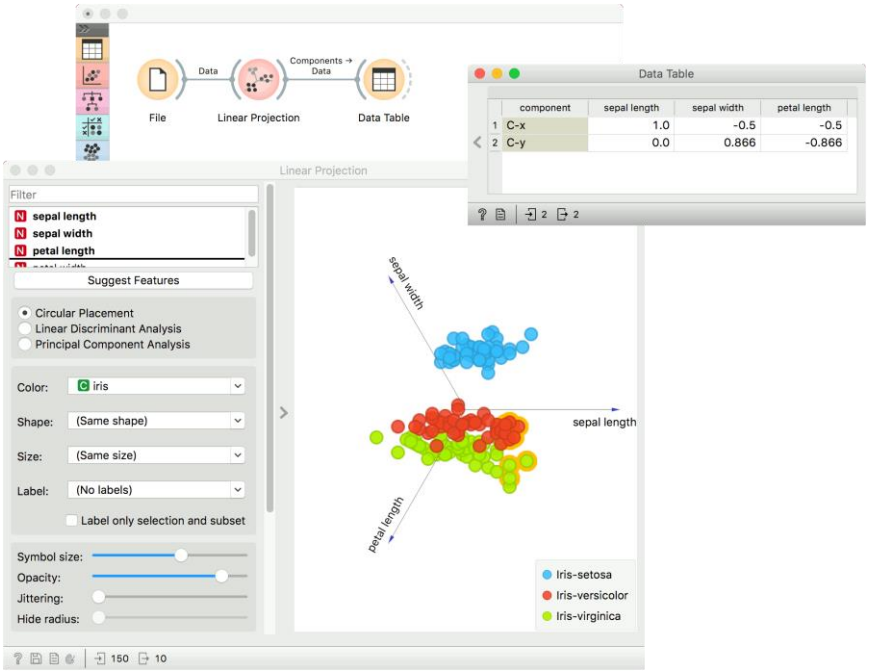


1. Axes in the projection that are displayed and other available axes. Optimize your projection by using Suggest Features. This feature scores attributes and returns the top scoring attributes with a simultaneous visualiza- tion update. Feature scoring computes the classification accuracy (for classification) or MSE (regression) of k-nearest neighbors classifier on the projected, two-dimensional data. The score reflects how well the classes in the projection are separated.
2. Choose the type of projection:
   - Circular Placement
   - Linear Discriminant Analysis
   - Principal Component Analysis
3. Set the color of the displayed points. Set shape, size, and label to differentiate between points. Label only selected points labels only selected data instances.
4. Adjust plot properties:
   - Symbol size: set the size of the points.
   - Opacity: set the transparency of the points.
   - Jittering: Randomly disperse points with jittering to prevent them from overlapping.
   - Hide radius: Axes inside the radius are hidden. Drag the slider to change the radius.
5. Additional plot properties:
   - Show color regions colors the graph by class.
   - Show legend displays a legend on the right. Click and drag the legend to move it.
6. Select, zoom, pan, and zoom to fit are the options for exploring the graph. The manual selection of data instances works as an angular/square selection tool. Double-click to move the projection. Scroll in or out for zoom.
7. If Send automatically is ticked, changes are communicated automatically. Alternatively, press Send.

Example:

The Linear Projection widget works just like other visualization widgets. Below, we connected it to the File widget to see the set projected on a 2-D plane. Then we selected the

data for further analysis and connected it to the Data Table widget to see the details of the selected subset.



8.9.3. References

Koren Y., Carmel L. (2003). Visualization of labeled data using linear transformations. In Proceedings of IEEE Information Visualization 2003, (InfoVis'03). Available here.

Boulesteix A.-L., Strimmer K. (2006). Partial least squares: a versatile tool for the analysis of high-dimensional ge- nomic data. Briefings in Bioinformatics, 8(1), 32-44. Abstract here.

Leban G., Zupan B., Vidmar G., Bratko I. (2006). VizRank: Data Visualization Guided by Machine Learning. Data Mining and Knowledge Discovery, 13, 119-136. Available here.

1. The larger the size of the data (although the number of columns and rows) makes it difficult to interpret and handle the data.
2. Data visualization refers to ways to represent data in different ways, such as pictures, charts, and so on.
3. Box plots can be based on the data's representative values (Q1, Q2, median, maximum, minimum, mean, standard deviation) and their distribution.
4. Violin plots can determine the distribution based on the value of the data. Depending on the characteristics, you can compare at a glance how much data is located in which data interval.
5. Distribution can tell the distribution of data based on each characteristic.
6. Heatmap expresses the relationship between each data in color, making it easy to understand at a glance.
7. Scatter plots are often used to express the correlation of data. Because it is represented in a plane based on two characteristics, it is easy to interpret the relationship between the two data.
8. Line plots are often used to compare changes in data in time series data. You can view and interpret the flow of data that changes over time or specific criteria.
9. Bar plot can be used to compare the size of the interval or discrete values of the data. Although it can be utilized similarly to line plots, line plots are mainly used to determine continuous value changes, and bar plots are used to compare discrete values.

1. Explain why you need data visualization
2. What are the representative values that can be found in the box plot?
3. Which plot is most appropriate to determine the correlation between data characteristics?
4. What is the difference between a bar plot and a line plot? Which data is more appropriate for each representation?

Exercises

1. Load iris data from the Datasets widget and find out what the characteristics of the data mean.
2. Express the number of data according to the flower variety in iris dataset as a bar plot.
3. Check the representative value of the petal width using the box plot and explain the characteristics of the data.
4. Analyse the correlation between the width and length of petals and the width and length of sepal with a scatter plot, and interpret the relationship according to the variety.