

Content Organization Series

Zheng Li

Database System Concepts

Content Overview

First Edition



| E-Book

Content Organization Series

Database System Concepts

Zheng Li

E-Book

Simple Introduction

Content organized from:

Database System Concepts(Seventh Edition)

Written by Abraham Silberschatz, Henry F. Korth and S. Sudarshan



Contents

1	Introduction	1
1.1	Database-System Applications	1
1.2	Purpose of Database Systems	1
1.3	View of Data	2
1.3.1	Data Models	2
1.3.2	Data Abstraction	2
1.3.3	Instances and Schemas	2
1.4	Database Languages	2
1.4.1	Data-Definition Language	3
1.4.2	Data-Manipulation Language	3
1.4.3	Database Access from Application Programs	3
1.5	Database Design	4
1.6	Database Engine	4
1.6.1	Storage Manager	4
1.6.2	The Query Processor	5
1.6.3	Transaction Management	5
1.7	Database and Application Architecture	6
1.8	Database Users and Administrators	6
1.8.1	Database Administrators	6
1.9	History of Database Systems	6
2	Introduction to the Relational Model	7
2.1	Structure of Relational Databases	7
2.2	Database Schema	7
2.3	Keys	7
2.4	Schema Diagrams	8
2.5	Relational Query Languages	8
2.6	The Relational Algebra	8
2.6.1	The Select Operation	9
2.6.2	The Project Operation	9
2.6.3	Composition of Relational Operations	9
2.6.4	The Cartesian-Product Operation	9
2.6.5	The Join Operation	9
2.6.6	Set Operations	9

2.6.7	The Assignment Operation	10
2.6.8	The Rename Operation	10

A **database-management system (DBMS)** is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the **database**, contains information relevant to an enterprise.

Sec 1.1 Database-System Applications

Broadly speaking, there are two modes in which databases are used.

- The first mode is to support **online transaction processing**, where a large number of users use the database, with each user retrieving relatively small amounts of data, and performing small updates.
- The second mode is to support **data analytics**, that is, the processing of data to draw conclusions, and infer rules or decision procedures, which are then used to drive business decisions.

Sec 1.2 Purpose of Database Systems

One way to keep the information on a computer is to store it in operating-system files.

This typical **file-processing system** is supported by a conventional operating system.

Keeping organizational information in a file-processing system has a number of major disadvantages:

- **Data redundancy and inconsistency.** Since different programmers create the files and application programs over a long period, the various files are likely to have different structures, and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). In addition, it may lead to **data inconsistency**; that is, the various copies of the same data may no longer agree.
- **Integrity problems.** The data values stored in the database must satisfy certain types of **consistency constraints**.

Sec 1.3 View of Data

1.3.1 Data Models

Underlying the structure of a database is the **data model**: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

The data models can be classified into four different categories:

- **Relational Model**. The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as **relations**.
- **Entity-Relationship Model**.
- **Semi-structured Data Model**.
- **Object-Based Data Model**.

1.3.2 Data Abstraction

Since many database-system users are not computer trained, developers hide the complexity from users through several levels of **data abstraction**, to simplify users' interactions with the system:

- **Physical level**.
- **Logical level**. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**.
- **View level**.

1.3.3 Instances and Schemas

The collection of information stored in the database at a particular moment is called an **instance** of the database. The overall design of the database is called the database **schema**.

The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called **subschemas**, that describe different views of the database.

Sec 1.4 Database Languages

A database system provides a **data-definition language (DDL)** to specify the database schema and a **data-manipulation language (DML)** to express database queries and updates.

1.4.1 Data-Definition Language

We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a **data storage and definition** language.

In general, a constraint can be an arbitrary predicate pertaining to the database. However, arbitrary predicates may be costly to test. Thus, database systems implement only those integrity constraints that can be tested with minimal overhead:

- **Domain Constraints.**
- **Referential Integrity.**
- **Authorization.** We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database. These differentiations are expressed in terms of **authorization**, the most common being: **read authorization**, which allows reading, but not modification, of data; **insert authorization**, which allows insertion of new data, but not modification of existing data; **update authorization**, which allows modification, but not deletion, of data; and **delete authorization**, which allows deletion of data.

The output of the DDL is placed in the **data dictionary**, which contains **metadata** – that is, data about data.

1.4.2 Data-Manipulation Language

A **data-manipulation language (DML)** is a language that enables users to access or manipulate as organized by the appropriate data model. There are basically two types of data-manipulation language:

- **Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.
- **Declarative DMLs** (also referred to as **nonprocedural DMLs**) require a user to specify *what* data are needed *without* specifying how to get those data.

A **query** is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a **query language**.

1.4.3 Database Access from Application Programs

Non-procedural query languages are not as powerful as a universal Turing machine; that is, there are some computations that are possible using a general-purpose programming language but are not possible using SQL. SQL also does not support actions such as input from users, output to displays, or communication over the network. Such computations and actions must be written in a *host* language. **Application programs** are programs that are used to interact with the database in this fashion.

Sec 1.5 Database Design

A high-level data model provides the database designer with a conceptual framework in which to specify the data requirements of the database users and how the database will be structured to fulfill these requirements. The initial phase of database design, then, is to characterize fully the data needs of the prospective database users.

Next, the designer chooses a data model, and by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database. The schema developed at this **conceptual-design** phase provides a detailed overview of the enterprise.

In terms of the relational model, the conceptual-design process involves decisions on *what* attributes we want to capture in the database and *how to group* these attributes to form the various tables. The "how" part is mainly a computer-science problem. There are principally two ways to tackle the problem. The first one is to use the entity-relationship model; the other is to employ a set of algorithms (collectively known as **normalization**) that takes as input the set of all attributes and generates a set of tables.

In a **specification of functional requirements**, users describe the kinds of operations (or transactions) that will be performed on the data.

In the **logical-design phase**, the designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used. The designer uses the resulting system-specific database schema in the subsequent **physical-design phase**, in which the physical features of the database are specified.

Sec 1.6 Database Engine

The functional components of a database system can be broadly divided into the storage manager, the **query processor** components, and the transaction management component.

1.6.1 Storage Manager

The **storage manager** is the component of a database system that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

The storage manager components include:

- **Authorization and integrity manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.
- **Transaction manager**, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicts.
- **File manager**, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

- **Buffer manager**, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory.

The storage manager implements several data structures as part of the physical system implementation:

- **Data files**, which store the database itself.
- **Data dictionary**, which stores metadata about the structure of the database, in particular the schema of the database.
- **Indices**, which can provide fast access to data items.

1.6.2 The Query Processor

The query processor components include:

- **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
- **DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query-evaluation engine understands.

A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs **query optimization**; that is, it picks the lowest cost evaluation plan from among the alternatives.

- **Query evaluation engine**, which executes low-level instructions generated by the DML compiler.

1.6.3 Transaction Management

Often, several operations on the database form a single logical unit of work. An example is a funds transfer in which one account *A* is debited and another account *B* is credited. Clearly, it is essential that either both the credit and debit occur, or that neither occur. This all-or-none requirement is called **atomicity**. In addition, it is essential that the execution of the funds transfer preserves the consistency of the database. This correctness requirement is called **consistency**. Finally, after the successful execution of a funds transfer, the new values of the balances of accounts *A* and *B* must persist, despite the possibility of system failure. This persistence requirement is called **durability**.

A **transaction** is a collection of operations that performs a single logical function in a database application.

Ensuring the atomicity and durability properties is the responsibility of the database system itself – specifically, of the **recovery manager**. If we are to ensure the atomicity property, a failed transaction must have no effect on the state of the database. Thus, the database must be restored to the state in which it was before the transaction in question started executing. The database system must therefore perform **failure recovery**, that is, it must detect system failures and restore the database to the state that existed prior to the occurrence of the failure.

It is the responsibility of the **concurrency-control manager** to control the interaction among the concurrent transactions, to ensure the consistency of the database. The **transaction**

manager consists of the concurrency-control manager and the recovery manager.

Sec 1.7 Database and Application Architecture

Earlier-generation database applications used a **two-tier architecture**, where the application resides at the client machine, and invokes database system functionality at the server machine through query language statements.

In contrast, modern database applications use a **three-tier architecture**, where the client machine acts as merely a front end and does not contain any direct database calls; web browsers and mobile applications are the most commonly used application clients today. The front end communicates with an **application server**. The application server, in turn, communicates with a database system to access data. The **business logic** of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients.

Sec 1.8 Database Users and Administrators

1.8.1 Database Administrators

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a **database administrators (DBA)**.

Sec 1.9 History of Database Systems

Techniques for data storage and processing have evolved over the years:

- **2000s:** In the latter part of the decade, the use of data analytics and **data mining** in enterprises became ubiquitous.

Sec 2.1 Structure of Relational Databases

A relational database consists of a collection of **tables**, each of which is assigned a unique name.

In mathematical terminology, a **tuple** is simply a sequence (or list) of values. A relationship between n values is represented mathematically by an **n -tuple** of values, that is, a tuple with n values, which corresponds to a row in a table.

Thus, in the relational model the term **relation** is used to refer to a table, while the term **tuple** is used to refer to a row. Similarly, the term **attribute** refers to a column of a table.

We use the term **relation instance** to refer to a specific instance of a relation, that is, containing a specific set of rows.

For each attribute of a relation, there is a set of permitted values, called the **domain** of that attribute.

A domain is **atomic** if elements of the domain are considered to be indivisible units.

The **null value** is a special value that signifies that the value is unknown or does not exist.

Sec 2.2 Database Schema

When we talk about a database, we must differentiate between the **database schema**, which is the logical design of the database, and the **database instance**, which is a snapshot of the data in the database at a given instant in time.

The concept of a relation corresponds to the programming-language notion of a variable, while the concept of a **relation schema** corresponds to the programming-language notion of type definition.

Sec 2.3 Keys

A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.

We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **candidate keys**.

We shall use the term **primary key** to denote a candidate key that is chosen by the database designer as the principal means of identifying tuples within a relation.

2.4. SCHEMA DIAGRAMS

The designation of a key represents a constraint in the real-world enterprise being modeled. Thus, primary keys are also referred to as **primary key constraints**.

A **foreign-key constraint** from attribute(s) A of relation r_1 to the primary-key B of relation r_2 states that on any database instance, the value of A for each tuple in r_1 must also be the value of B for some tuple in r_2 . Attribute set A is called a **foreign key** from r_1 , referencing r_2 . The relation r_1 is also called the **referencing relation** of the foreign-key constraint, and r_2 is called the **referenced relation**.

In general, a **referential integrity constraint** requires that the values appearing in specified attributes of any tuple in the referencing relation also appear in specified attributes of at least one tuple in the referenced relation.

Sec 2.4 Schema Diagrams

A database schema, along with primary key and foreign-key constraints, can be depicted by **schema diagrams**.

Sec 2.5 Relational Query Languages

A **query language** is a language in which a user requests information from the database. In an **imperative query language**, the user instructs the system to perform a specific sequence of operations on the database to compute the desired result; such languages usually have a notion of state variables, which are updated in the course of the computation.

In a **functional query language**, the computation is expressed as the evaluation of functions that may operate on data in the database or on the results of other functions; functions are side-effect free, and they do not update the program state.¹ In a **declarative query language**, the user describes the desired information without giving a specific sequence of steps or function calls for obtaining that information; the desired information is typically described using some form of mathematical logic.

There are a number of "pure" query languages.

- The *relational algebra* is a functional query language.
- The tuple relational calculus and domain relational calculus are declarative.

Sec 2.6 The Relational Algebra

The relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result.

¹The term *procedure language* include functional languages; however, the term is also widely used to refer to imperative languages.

Some of these operations are called *unary* operations because they operate on one relation. The other operations operate on pairs of relations and are, therefore, called *binary* operations.

2.6.1 The Select Operation

The **select** operation selects tuples that satisfy a given predicate.

2.6.2 The Project Operation

Suppose we want to list all instructors' *ID*, *name*, and *salary*, but we do not care about the *dept_name*. The **project** operation allows us to produce this relation.

2.6.3 Composition of Relational Operations

In general, since the result of a relational-algebra operation is of the same type (relation) as its inputs, relational-algebra operations can be composed together into a **relational-algebra expression**.

2.6.4 The Cartesian-Product Operation

The **Cartesian-product** operation, denoted by a cross (\times), allows us to combine information from any two relations.

2.6.5 The Join Operation

Consider relations $r(R)$ and $s(S)$, and let θ be a predicate on attributes in the schema $R \cup S$. The **join** operation $r \bowtie_{\theta} s$ is defined as follows:

$$r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$$

2.6.6 Set Operations

In general, for a union operation to make sense:

1. We must ensure that the input relations to the union operation have the same number of attributes; the number of attributes of a relation is referred to as its **arity**.
2. When the attributes have associated types, the types of the i th attributes of both input relations must be the same, for each i .

Such relations are referred to as **compatible** relations.

The **intersection** operation, denote by \cap , allows us to find tuples that are in both the input relations.

The **set-difference** operation, denoted by $-$, allows us to find tuples that are in one relation but are not in another.

2.6.7 The Assignment Operation

The **assignment** operation, denoted by \leftarrow , works like assignment in a programming language.

2.6.8 The Rename Operation

Unlike relations in the database, the results of relational-algebra expressions do not have a name that we can use to refer to them. It is useful in some cases to give them names; the **rename** operator, denoted by the lowercase Greek letter rho (ρ), lets us to this.

Learning is all you need

