1. Data models, Schemas, and Instances
One fundamental characteristic of the database approach is that it provides some level of data abstraction by hiding details of data storage that are not needed by most database users. A data model - a collection of concepts that can be used to describe the structure of a database - provides the necessary means to achieve this abstraction. By structure of a database we mean the data types, relationships, and constraints that should hold on the data. Most data models also include a set of basic operations for specifying retrievals and updates on the database.

1.1 Categories of Data Models
Many data models have been proposed, and we can categorize them according to the types of concepts they use to describe the database structure.
1. Conceptual (high-level) data models provide concepts that are close to the way many end users perceive data. Conceptual Data Models use concepts such as entities, attributes, and relationships.
2. Physical Data Models describes how data is stored in the computer by representing information such as stored record formats, record orderings, and access paths. An access path is a structure that makes the search for particular database records efficient.
3. Implementation (record-oriented) data models: Provide concepts that fall between the above two, balancing user views with some computer storage details.

1.2 Schemas, Instances, and Database State
In any data model it is important to distinguish between the description of the database and the database itself. The description of a database is called the database schema, which is specified during database design and is not expected to change frequently. Most data models have certain conventions for displaying the schemas as diagrams. A displayed schema is called a schema diagram. Figure 7 shows a sample schema diagram. Diagram displays only the structure of each record type but not the actual instances of records.
Schema diagram displays only some aspect of a schema, such as the names of record types and data items, and some types of constraints.

<table>
<thead>
<tr>
<th>SUPPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuppNumber</td>
</tr>
<tr>
<td>SuppName</td>
</tr>
<tr>
<td>Status</td>
</tr>
<tr>
<td>City</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PART</th>
</tr>
</thead>
<tbody>
<tr>
<td>PartNumber</td>
</tr>
<tr>
<td>PartName</td>
</tr>
<tr>
<td>Color</td>
</tr>
<tr>
<td>Weight</td>
</tr>
<tr>
<td>City</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DELIVERY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuppNumber</td>
</tr>
<tr>
<td>PartNumber</td>
</tr>
<tr>
<td>Quantity</td>
</tr>
</tbody>
</table>

Figure 7: Sample schema diagram

The actual data in a database may change quite frequently; for example, the database shown in Figure 7 changes every time we add a supplier or enter a new color for a part. The data in the
The database in a particular moment in time is called database state or snapshot. It is also called the current set of occurrences or instances in the database. Every time we insert or delete a record, or change the value of a data item in a record, we change a one state of the database into another state.

The distinction between database schema and database state is very important. When we define a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the empty state with no data. We get the initial state of the database when the database is first loaded with the initial data. From then, on every time an update operation is applied to the database, we get another database state. At any point in time, the database has a current state. The DBMS is partly responsible for ensuring that every state of the database is a valid state - that is, a state that satisfies the structure and constraints specified in the schema. Example in Figure 8 shows current state of the database.

![Database Tables](image)

**Figure 8: Sample current state of database**

2. **DBMS Architecture**

Database management systems are complex software which were often developed and optimized over years. From the view of the user, however, most of them have a quite similar basic architecture. The discussion of this basic architecture shall help to understand the connection with data modeling and the introductionally to this module postulated 'data independence' of the database approach.

2.1 **The Three-Schema Architecture (Figure 9)**

Three important characteristics of the database approach are insulation of programs and data, support of multiple user views and use of a catalog to store the database description. Let’s specify architecture for database systems, called the three-schema architecture, which was proposed to help achieve and visualize these characteristics.

The goal of the three-schema architecture is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels:

- **The internal level** has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
• **The conceptual level** has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. A high-level data model or an implementation data model can be used at this level.

• **The external or view level** includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model can be used at this level.

![Three-Schemes Architecture](image)

**Figure 9.** Three-Schemes Architecture

## 2.2 Data Independence

The three-schema architecture can be used to explain the **concept of data independence**, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define **two types of data independence**:

• **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), or to reduce the database (by removing a record type or data item). In the latter case, external schemas that refer only to the remaining data should not be affected. Only the view definition and the mappings need be changed in a DBMS that supports logical data independence. Application programs that reference the external schema constructs must work as before, after the conceptual schema undergoes a logical reorganization. Changes to constraints can be applied also to the conceptual schema without affecting the external schemas or application programs.

• **Physical data independence** is the capacity to change the internal schema without having to change the conceptual (or external) schemas. Changes to the internal schema may be needed because some **physical files had to be reorganized - for example**, by creating additional access structures - to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.
3. Database Languages and Database Interfaces
In this unit, it is explained how a data models gets into a database system and how the information gets to the users. More correctly formulated the following questions will be answered:

- How does an application interact with a database management system?
- How does a user look at a database system?
- How can a user query a database system and view the results in his/her application?

3.1. Database Languages

**DDL**
For describing data and data structures a suitable description tool, a data definition language (DDL), is needed. With this help a data scheme can be defined and also changed later. Typical DDL operations are:

- Creation of tables and definition of attributes (CREATE TABLE ...)
- Change of tables by adding or deleting attributes (ALTER TABLE ...)
- Deletion of whole table including content (!) (DROP TABLE ...)

**DML**
Additionally a language for the descriptions of the operations with data like store, search, read, change, etc. the so-called data manipulation, is needed. Such operations can be done with a data manipulation language (DML). Within such languages keywords like insert, modify, update, delete, select, etc. are common. Typical DML operations are:

- Add data (INSERT)
- Change data (UPDATE)
- Delete data (DELETE)
- Query data (SELECT)

Often these two languages for the definition and manipulation of databases are combined in one comprehensive language.

3.2. Database Interfaces (Figure 10)

![Figure 10. Working Principle of a Database Interface](image-url)
The application poses with the help of SQL, a query language, a query to the database system. There, the corresponding answer (result set) is prepared and also with the help of SQL given back to the application. This communication can take place interactively or be embedded into another language.

**Type and Use of the Database Interface**

Following, two important uses of a database interface like SQL are listed:

**Interactive**: SQL can be used interactively from a terminal.

**Embedded**: SQL can be embedded into another language (host language) which might be used to create a database application.

**3.3. User Interfaces**

A user interface is the view of a database interface that is seen by the user. User interfaces are often graphical or at least partly graphical (GUI - graphical user interface) constructed and offer tools which make the interaction with the database easier.

**1. Form-based Interfaces (Figure 11)**

This interface consists of forms which are adapted to the user. He/She can fill in all of the fields and make new entries to the database or only some of the fields to query the other ones. But some operations might be restricted by the application. Form-based user interfaces are widespread and are a very important means of interacting with a DBMS. They are easy to use and have the advantage that the user does not need special knowledge about database languages like SQL.

![Figure 11. Example of a Form-based User Interface](image-url)
2. Text-based Interfaces (Figure 12)
To be able to administrate the database or for other professional users there are possibilities to communicate with the DBMS directly in the query language (in code form) via an input/output window.
Text-based interfaces are very powerful tools and allow a comprehensive interaction with a DBMS. However, the use of these is based on active knowledge of the respective database language.

Figure 12. Example of a Text-base User Interface