DATABASE III

3.1 DEFINITION A **database** is an organized collection of data. The data are typically organized to model aspects of reality in a way that supports processes requiring information. For example, modelling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

3.2 Design and modeling

The first task of a database designer is to produce a conceptual data model that reflects the structure of the information to be held in the database. A common approach to this is to develop an entity-relationship model, often with the aid of drawing tools. Another popular approach is the Unified Modeling Language. A successful data model will accurately reflect the possible state of the external world being modeled: for example, if people can have more than one phone number, it will allow this information to be captured. Designing a good conceptual data model requires a good understanding of the application domain; it typically involves asking deep questions about the things of interest to an organisation, like "can a customer also be a supplier?", or "if a product is sold with two different forms of packaging, are those the same product or different products?", or "if a plane flies from New York to Dubai via Frankfurt, is that one flight or two (or maybe even three)?". The answers to these questions establish definitions of the terminology used for entities (customers, products, flights, flight segments) and their relationships and attributes.

Producing the conceptual data model sometimes involves input from business processes, or the analysis of workflow in the organization. This can help to establish what information is needed in the database, and what can be left out. For example, it can help when deciding whether the database needs to hold historic data as well as current data.

Having produced a conceptual data model that users are happy with, the next stage is to translate this into a schema that implements the relevant data structures within the database. This process is often called logical database design, and the output is a logical data model expressed in the form of a schema. Whereas the conceptual data model is (in theory at least) independent of the choice of database technology, the logical data model will be expressed in terms of a particular database model supported by the chosen DBMS. (The terms *data model* and *database model* are often used interchangeably, but in this article we use *data model* for the design of a

specific database, and *database model* for the modelling notation used to express that design.)

The most popular database model for general-purpose databases is the relational model, or more precisely, the relational model as represented by the SQL language. The process of creating a logical database design using this model uses a methodical approach known as normalization. The goal of normalization is to ensure that each elementary "fact" is only recorded in one place, so that insertions, updates, and deletions automatically maintain consistency.

The final stage of database design is to make the decisions that affect performance, scalability, recovery, security, and the like. This is often called *physical database design*. A key goal during this stage is data independence, meaning that the decisions made for performance optimization purposes should be invisible to end-users and applications. Physical design is driven mainly by performance requirements, and requires a good knowledge of the expected workload and access patterns, and a deep understanding of the features offered by the chosen DBMS.

Another aspect of physical database design is security. It involves both defining access control to database objects as well as defining security levels and methods for the data itself.

Models

A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated. The most popular example of a database model is the relational model (or the SQL approximation of relational), which uses a table-based format.

Common logical data models for databases include:

- Hierarchical database model
- Network model
- Relational model
- Entity–relationship model
    - Enhanced entity–relationship model
- Object model
- Document model
- Entity–attribute–value model
- Star schema

An object-relational database combines the two related structures.

Physical data models include:

- Inverted index
- Flat file

Other models include:

- Associative model
- Multidimensional model
- Multivalue model
- Semantic model
- XML database

3.3 External, conceptual, and internal views

A database management system provides three views of the database data:

- The **external level** defines how each group of end-users sees the organization of data in the database. A single database can have any number of views at the external level.
- The **conceptual level** unifies the various external views into a compatible global view. It provides the synthesis of all the external views. It is out of the scope of the various database end-users, and is rather of interest to database application developers and database administrators.
- The **internal level** (or *physical level*) is the internal organization of data inside a DBMS (see Implementation section below). It is concerned with cost, performance, scalability and other operational matters. It deals with storage layout of the data, using storage structures such as indexes to enhance performance. Occasionally it stores data of individual views (materialized views), computed from generic data, if performance justification exists for such redundancy. It balances all the external views' performance requirements, possibly conflicting, in an attempt to optimize overall performance across all activities.

While there is typically only one conceptual (or logical) and physical (or internal) view of the data, there can be any number of different external views. This allows users to see database information in a more business-related way rather than from a technical, processing viewpoint. For example, a financial department of a company needs the payment details of all employees as part of the company's expenses, but

does not need details about employees that are the interest of the human resources department. Thus different departments need different *views* of the company's database.

The three-level database architecture relates to the concept of *data independence* which was one of the major initial driving forces of the relational model. The idea is that changes made at a certain level do not affect the view at a higher level. For example, changes in the internal level do not affect application programs written using conceptual level interfaces, which reduces the impact of making physical changes to improve performance.

The conceptual view provides a level of indirection between internal and external. On one hand it provides a common view of the database, independent of different external view structures, and on the other hand it abstracts away details of how the data is stored or managed (internal level). In principle every level, and even every external view, can be presented by a different data model. In practice usually a given DBMS uses the same data model for both the external and the conceptual levels (e.g., relational model). The internal level, which is hidden inside the DBMS and depends on its implementation (see Implementation section below), requires a different level of detail and uses its own types of data structure types.

Separating the *external*, *conceptual* and *internal* levels was a major feature of the relational database model implementations that dominate 21st century databases.

Languages

Database languages are special-purpose languages, which do one or more of the following:

- Data definition language – defines data types and the relationships among them
- Data manipulation language – performs tasks such as inserting, updating, or deleting data occurrences
- Query language – allows searching for information and computing derived information

Database languages are specific to a particular data model. Notable examples include:

- SQL combines the roles of data definition, data manipulation, and query in a single language. It was one of the first commercial languages for the

relational model, although it departs in some respects from the relational model as described by Codd (for example, the rows and columns of a table can be ordered). SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. The standards have been regularly enhanced since and is supported (with varying degrees of conformance) by all mainstream commercial relational DBMSs.

- OQL is an object model language standard (from the Object Data Management Group). It has influenced the design of some of the newer query languages like JDOQL and EJB QL.

- XQuery is a standard XML query language implemented by XML database systems such as MarkLogic and eXist, by relational databases with XML capability such as Oracle and DB2, and also by in-memory XML processors such as Saxon.

A database language may also incorporate features like:

- DBMS-specific Configuration and storage engine management
- Computations to modify query results, like counting, summing, averaging, sorting, grouping, and cross-referencing
- Constraint enforcement (e.g. in an automotive database, only allowing one engine type per car)
- Application programming interface version of the query language, for programmer convenience

Performance, security, and availability

Because of the critical importance of database technology to the smooth running of an enterprise, database systems include complex mechanisms to deliver the required performance, security, and availability, and allow database administrators to control the use of these features.

3.4 Storage

Database storage is the container of the physical materialization of a database. It comprises the *internal* (physical) *level* in the database architecture. It also contains all the information needed (e.g., metadata, "data about the data", and internal data structures) to reconstruct the *conceptual level* and *external level* from the internal level when needed. Putting data into permanent storage is generally the

responsibility of the database engine a.k.a. "storage engine". Though typically accessed by a DBMS through the underlying operating system (and often utilizing the operating systems' file systems as intermediates for storage layout), storage properties and configuration setting are extremely important for the efficient operation of the DBMS, and thus are closely maintained by database administrators. A DBMS, while in operation, always has its database residing in several types of storage (e.g., memory and external storage). The database data and the additional needed information, possibly in very large amounts, are coded into bits. Data typically reside in the storage in structures that look completely different from the way the data look in the conceptual and external levels, but in ways that attempt to optimize (the best possible) these levels' reconstruction when needed by users and programs, as well as for computing additional types of needed information from the data (e.g., when querying the database).

Some DBMSs support specifying which character encoding was used to store data, so multiple encodings can be used in the same database.

Various low-level database storage structures are used by the storage engine to serialize the data model so it can be written to the medium of choice. Techniques such as indexing may be used to improve performance. Conventional storage is row-oriented, but there are also column-oriented and correlation databases.

Materialized views

Often storage redundancy is employed to increase performance. A common example is storing *materialized views*, which consist of frequently needed *external views* or query results. Storing such views saves the expensive computing of them each time they are needed. The downsides of materialized views are the overhead incurred when updating them to keep them synchronized with their original updated database data, and the cost of storage redundancy.

Replication

Occasionally a database employs storage redundancy by database objects replication (with one or more copies) to increase data availability (both to improve performance of simultaneous multiple end-user accesses to a same database object, and to provide resiliency in a case of partial failure of a distributed database). Updates of a replicated object need to be synchronized across the object copies. In many cases the entire database is replicated.

Security

Database security deals with all various aspects of protecting the database content, its owners, and its users. It ranges from protection from intentional unauthorized database uses to unintentional database accesses by unauthorized entities (e.g., a person or a computer program).

Database access control deals with controlling who (a person or a certain computer program) is allowed to access what information in the database. The information may comprise specific database objects (e.g., record types, specific records, data structures), certain computations over certain objects (e.g., query types, or specific queries), or utilizing specific access paths to the former (e.g., using specific indexes or other data structures to access information). Database access controls are set by special authorized (by the database owner) personnel that uses dedicated protected security DBMS interfaces.

This may be managed directly on an individual basis, or by the assignment of individuals and privileges to groups, or (in the most elaborate models) through the assignment of individuals and groups to roles which are then granted entitlements. Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called "subschemas". For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data. If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases.

Data security in general deals with protecting specific chunks of data, both physically (i.e., from corruption, or destruction, or removal; e.g., see physical security), or the interpretation of them, or parts of them to meaningful information (e.g., by looking at the strings of bits that they comprise, concluding specific valid credit-card numbers; e.g., see data encryption).

Change and access logging records who accessed which attributes, what was changed, and when it was changed. Logging services allow for a forensic database audit later by keeping a record of access occurrences and changes. Sometimes application-level code is used to record changes rather than leaving this to the database. Monitoring can be set up to attempt to detect security breaches.

Transactions and concurrency

Database transactions can be used to introduce some level of fault tolerance and data integrity after recovery from a crash. A database transaction is a unit of work, typically encapsulating a number of operations over a database (e.g., reading a database object, writing, acquiring lock, etc.), an abstraction supported in database and also other systems. Each transaction has well defined boundaries in terms of which program/code executions are included in that transaction (determined by the transaction's programmer via special transaction commands).

The acronym ACID describes some ideal properties of a database transaction: Atomicity, Consistency, Isolation, and Durability.

Migration

A database built with one DBMS is not portable to another DBMS (i.e., the other DBMS cannot run it). However, in some situations it is desirable to move, migrate a database from one DBMS to another. The reasons are primarily economical (different DBMSs may have different total costs of ownership or TCOs), functional, and operational (different DBMSs may have different capabilities). The migration involves the database's transformation from one DBMS type to another. The transformation should maintain (if possible) the database related application (i.e., all related application programs) intact. Thus, the database's conceptual and external architectural levels should be maintained in the transformation. It may be desired that also some aspects of the architecture internal level are maintained. A complex or large database migration may be a complicated and costly (one-time) project by itself, which should be factored into the decision to migrate. This in spite of the fact that tools may exist to help migration between specific DBMSs. Typically a DBMS vendor provides tools to help importing databases from other popular DBMSs.

Building, maintaining, and tuning

After designing a database for an application, the next stage is building the database. Typically an appropriate general-purpose DBMS can be selected to be utilized for this purpose. A DBMS provides the needed user interfaces to be utilized by database administrators to define the needed application's data structures within the DBMS's respective data model. Other user interfaces are used to select needed DBMS parameters (like security related, storage allocation parameters, etc.).

When the database is ready (all its data structures and other needed components are defined) it is typically populated with initial application's data (database

initialization, which is typically a distinct project; in many cases using specialized DBMS interfaces that support bulk insertion) before making it operational. In some cases the database becomes operational while empty of application data, and data is accumulated during its operation.

After the database is created, initialised and populated it needs to be maintained. Various database parameters may need changing and the database may need to be tuned (tuning) for better performance; application's data structures may be changed or added, new related application programs may be written to add to the application's functionality, etc.

Backup and restore

Sometimes it is desired to bring a database back to a previous state (for many reasons, e.g., cases when the database is found corrupted due to a software error, or if it has been updated with erroneous data). To achieve this a **backup** operation is done occasionally or continuously, where each desired database state (i.e., the values of its data and their embedding in database's data structures) is kept within dedicated backup files (many techniques exist to do this effectively). When this state is needed, i.e., when it is decided by a database administrator to bring the database back to this state (e.g., by specifying this state by a desired point in time when the database was in this state), these files are utilized to **restore** that state.

Other

Other DBMS features might include:

- Database logs
- Graphics component for producing graphs and charts, especially in a data warehouse system
- **Query optimizer** – Performs query optimization on every query to choose for it the most efficient *query plan* (a partial order (tree) of operations) to be executed to compute the query result. May be specific to a particular storage engine.
- Tools or hooks for database design, application programming, application program maintenance, database performance analysis and monitoring, database configuration monitoring, DBMS hardware configuration (a DBMS and related database may span computers, networks, and storage units) and related database mapping (especially for a distributed DBMS), storage allocation and database layout monitoring, storage migration, etc.