THREE SCHEMA ARCHITECTURE OF A DATABASE SYSTEM

*Architecture of a system means its **structural design** i.e. a description of the components present in it and their interconnection.*

BACKGROUND

An early proposal for a standard terminology and general architecture for database systems was produced in 1971 by the DBTG (Data Base Task Group) appointed by the *Conference on Data Systems and Languages* (CODASYL, 1971). The DBTG recognized the need for a two-level approach with a system view called the ***schema*** and user views called ***subschemas***.

In late 1972, the Computer and Information Processing Committee (X3) of the American National Standards Institute (ANSI) established a Study Group on Database Management Systems under the auspices of its Standards Planning and Requirements Committee (SPARC). The mission of the study group was to study the feasibility of setting up standards in this area as well as determining which aspects should be standardized if it was feasible. The study group issued its interim report in 1975 and its final report in 1977. The architectural framework proposed in these reports came to be known as the "ANSI/SPARC architecture" its full title being "ANSI/X3/SPARC DBMS Framework". The study group proposed that the interfaces be standardized and defined an architectural framework that contained 43 interfaces, 14 of which would deal with the physical storage subsystem of the computer and therefore not be considered essential parts of the DBMS architecture.

ANSI/SPARC recognized the need for a three-level approach with a system catalog. These proposals reflected those published by the *IBM user organizations Guide and Share* some years previously and concentrated on the need for an implementation-independent layer to isolate programs from underlying representational issues (Guide/Share, 1970). The ANSI-SPARC model did not become an official standard. Nevertheless, it is considered the defacto industry standard for guiding the development of database systems and is the basis for understanding some of the functionality of a DBMS.

OBJECTIVES

- The Database Management Systems were rapidly adopted because of their ability to provide data to organizations in a controlled environment. This

control leads to minimum interaction between the way the user understands the data and the way it is physically stored. This is called ***Data Independence***. The three-level architecture provides a basis for achieving the data independence objective. It is basically identical to the architecture proposed by the ANSI/SPARC Study Group on Database Management Systems. The DBA should be able to change the database storage structures without affecting the users' view. The DBA should also be able to change the conceptual (or logical) structure of the database without affecting all users.

- In order for the database system to be usable, data must be retrieved efficiently. This concern has led to the design of complex data structures for the representation of data in the database and efficient access techniques such as indexing or hashing. Since many database systems' users are not computer trained so a major purpose of a DBMS is to provide them with an ***abstract view*** of the data by hiding certain details of how data is stored and manipulated. The complexity is hidden from them through several levels of abstraction in order to simplify their interaction with the system. Therefore the starting point for the design of a database must be an abstract and general description of the information requirements of the organization.
- Different users should be able to access the same data but have a different customized view of the data. Each user should be able to change the way he views the data and this change should not affect others users.
- An important objective of presenting this architecture is to provide a framework that will be useful for describing general database concepts and for explaining the structure of specific database systems.

These considerations led to the idea that there were at least three kinds of schema needed by an organization. The distinction between the logical and physical representation of data was officially recognized in 1978 when the ANSI/SPARC committee proposed a *generalized framework for database systems* (Tsichritzis and Klug, 1978). This framework provided a three-level architecture; three levels of abstraction at which a database could be viewed. A DBMS that provides these three levels of data is said to follow the *three-schema architecture*. The three levels in this model (called the ANSI/SPARC model) are the *conceptual level*, the *external levels (or views)* and the *internal level*. Separate data definition languages are often used to express

each of these types of schemas. Information about the conceptual, external and internal schemas is stored in the system catalog.

In order to illustrate the three-schema architecture, we will consider the example of a company that manufactures high quality, all-wood furniture and distributes it to stores in a metropolitan area. The company is organized into separate departments: Sales, Orders, Accounting, Manufacturing, Purchasing. Suppose that the company decides to develop and implement one or more databases which store data about customers, orders, invoice, product, work order, Raw Material, Vendor etc. There will be a need to provide different views of the data for different users. For example, a sales person will require views of customers and sales data while an accountant will require views of product and invoice data.

Fig 1 below depicts the relationship between these 3 views of a database. It is important to keep in mind that these are all views or models of the same organizational database, i.e., each organizational database has one physical and one conceptual schema and one or more user views. Thus the three schema architecture defines one database with multiple ways to look at this one set of data.

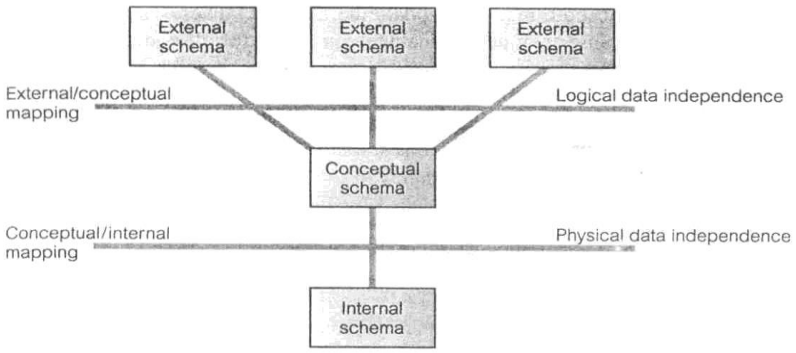

Fig 1: The Three Levels of Architecture

CONCEPTUAL LEVEL

- Also known as the *community logic level* or sometimes just the *logic level*, unqualified.
- At the conceptual level, we have an integrated view of the database called *conceptual view*.
- The conceptual level is the level of indirection between the other two levels.
- Provides both the mapping and the desired independence between the external and internal levels.
- In the relational model, the *base tables* correspond to the ANSI/SPARC conceptual level.
- The conceptual level is the level at which conceptual database design is performed (discussed ahead). Conceptual database design involves analysis of users' information needs and definition of data items needed to meet them. The result of conceptual design is the conceptual schema, a single, logical description of all data elements and their relationships.

Conceptual View

- *The conceptual view is a representation of the entire information content of the database in a form that is somewhat abstract in comparison with the way in which the data is physically stored.* It will also be quite different in general, from the way in which the data is viewed by any particular user.
- The conceptual view is the view of the data & database administrator who must decide what information is to be kept in the database.
- Here the entire database is described in terms of a number of relatively simple structures. Although implementation of simple structures at the conceptual level may involve complex physical level structures, the user of the conceptual level need not be aware of this.
- The conceptual view consists of many occurrences of each of many types of conceptual records. For example, it might consist of a collection of department record occurrences plus a collection of employee record occurrences plus a collection of supplier record occurrences plus a collection of part record occurrences (etc., etc.). A conceptual record is not necessarily the same as either an external record on the one hand, or a stored record on the other.

- Broadly speaking, the conceptual view is intended to be a view of the data *as it really is* rather than as users are forced to see it by the limitations of (for example) the particular language or hardware they might be using.
- There will be precisely one conceptual view of a database.

Conceptual Schema

Introduction

- The conceptual view is defined by means of the *conceptual schema*.
- A conceptual schema is a detailed and comprehensive specification of the overall structure of organizational data, while being independent of any database management technology.
- More specifically, a conceptual schema defines the whole database without reference to how data are stored in a computer's secondary memory.
- The conceptual schema evolves overtime – that is new data definitions are added to it as the database grows and matures.

What comprises conceptual schema?

The conceptual schema includes following requirements:-

- All the entities, their attributes and their relationships.
- It also includes definitions of each of the various conceptual record types that represent the entities, the data item types that represent their attributes and the relationships that will be stored.
- Integrity constraints,
- Semantic information about the data meanings and
- Security and Integrity information
- Some authorities would go so far as to suggest that the ultimate objective of the conceptual schema is to describe the complete enterprise – not just its data per se, but also how that data is used: how it flows from point to point within the enterprise, what it is used for at each point, what audit or other controls are to be applied at each point and so on.

Representing Conceptual Schema

i. A database developer will usually depict a conceptual schema in graphical format using Entity-Relationship or Object Modeling notations; this type of conceptual schema is usually called a data model. In addition, the specifications for the conceptual schema are stored as metadata in a repository or data dictionary [McFadden].

ii. The conceptual schema is written using another data definition language, the *conceptual DDL*. It is written in DDL, compiled by the DBMS, and stored in object form in the data dictionary and in source form as documentation. If physical data independence is to be achieved, then those conceptual DDL definitions must not involve any considerations of physical representation or access technique at all – they must be definitions of information content only. Thus there must be no reference in the conceptual schema to stored field representation, stored record sequence, indexes, hashing schemes, pointers or any other storage and access details. For instance, the description of an entity should contain only data types of attributes (for example, integer, real, character) and their length (such as the maximum number of digits or characters), but not any storage considerations, such as the number of bytes occupied. If the conceptual schema is made truly data-independent in this way, then the external schemas, which are defined in terms of the conceptual schema will a fortiori be data independent too.

At the conceptual level, the underlying data model may be *hierarchical*, *network, relational, ER-Model* etc. Ideally, the data model should make no reference to how the data model is implemented (inverted list, multiple linked lists and so on) or how the data are physically stored or accessed.

Guidelines for designing Conceptual Schema

- The conceptual view supports the external views, in that any data available to any user must be present in or derivable from the conceptual view. The conceptual model is relatively constant. When the DBA originally designs it, he or she tries to determine present and future information needs and attempts to develop a lasting model of the organization. Therefore, as new data needs arise, the conceptual model may already contain the objects required. If that is not the case, the DBA expands the conceptual model to include the new objects. A good conceptual model will be able to accommodate this change while still supporting

the old external views. Only the users who need access to the new data should be affected by the change.

- With a conceptual data model, we need to be able to consolidate user views, check for consistency (e.g. do all users refer to the same data item by the same name?) and validate that all data and relationships have been identified.
- It is important at this level to capture the semantics about data. For example, one user may mean all customer orders when she says "orders" but another may use this term only to refer to orders that still have products to be delivered.
- Further, information about who can do what with data must be captured and represented for subsequent use in developing the database.
- Finally, the conceptual model, since it is the basis for the design of the internal database, should be supplemented with information about database usage and maintenance so that an efficient internal structure can be devised.

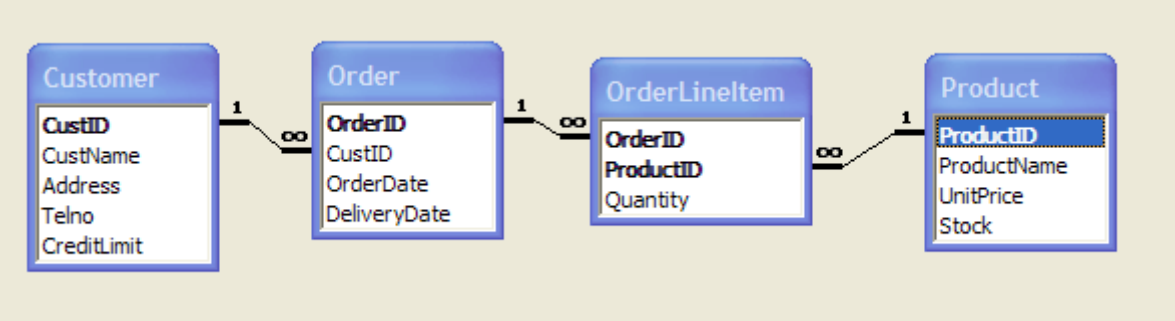

Fig 2: Conceptual Schema using ER-Model

The conceptual data model at Pine Valley Furniture company is shown in figure 2. Arrows between entities represent relationships between those entities (an arrow with two heads means a *many* relationship). A relationship is a logical association between those entities. For example, the arrow from Customer to Order in the figure indicates that a given customer may have several outstanding orders with Pine Valley Furniture at any given time. Also the arrow from Order to Invoice indicates that each customer order may have one or more invoices, each representing a partial shipment.

Figure 3 shows a sample relational schema for this database. The relations are defined by means of CREATE TABLE commands. The attributes in each table are named in the command and their types and lengths are specified. The schema is user-oriented and contains no information about physical implementation.

CREATE TABLE CUSTOMER (CUSTID NUMBER(4) **PRIMARY KEY**, CUSTNAME VARCHAR2(25) NOT NULL, ADDRESS VARCHAR2(50), TELNO VARCHAR2(20), CREDITLIMIT NUMBER(6) **CHECK** **(CREDITLIMIT BETWEEN 0 AND 100000)**);	CREATE TABLE ORDER (ORDERID NUMBER(4) **PRIMARY KEY**, CUSTID NUMBER(4), ORDERDATE DATE, DELIVERYDATE DATE, **FOREIGN KEY (CUSTID)** **REFERENCES CUSTOMER (CUSTID)**);
CREATE TABLE ORDERLINEITEM (ORDERID NUMBER(4), PRODUCTID NUMBER(4), QUANTITY NUMBER(5), **PRIMARY KEY (ORDERID, PRODUCTID),** **FOREIGN KEY (ORDERID)** **REFERENCES ORDER(ORDERID),** **FOREIGN KEY (PRODUCTID)** **REFERENCES PRODUCT(PRODUCTID));**	CREATE TABLE PRODUCT (PRODUCTID NUMBER(4) **PRIMARY KEY**, PRODUCTNAME VARCHAR2(25) NOT NULL, UNITPRICE NUMBER(4), STOCK NUMBER(4) **CHECK** **(STOCK BETWEEN 0 AND 1000)**);

Fig 3: Conceptual Schema using DDL

EXTERNAL LEVEL

It is the database structural level defining user views. External level, also known as the user logical level, is the one closest to the user i.e. it is the one concerned with the way the data is seen by individual users. It comprises several ***views*** of individual users or application programmers. Each view describes only part of the entire database and is derived from the conceptual view. Other entities, attributes or relationships that are not of interest may be represented in the database but the user will be unaware of them.

Despite the use of simple structures at the conceptual level, some complexity remains because of the larger size of the database. Many users of the database system will not be concerned with all this information. Each user has a model of the real world represented in a form that is suitable for that user. A particular user interacts with only certain entities in the real world and is interested in only some of their attributes and relationships.

Instead, such users need only a part of the database. To simplify their interaction with the system, the view level of abstraction is defined. The system may provide many external views for the same database.

Besides including different entities, attributes and relationships, different views may have different representations of the same data. For example, one user may believe dates are stored in the form (month, day, year), while another may believe they are represented as (year, month, day). Some views might include **virtual or calculated** data, data not actually stored as such, but created when needed. For example, *age* may not be actually stored, but *date of birth* may be, and *age* may be calculated by the system when the user refers to it. Views may even include data combined or calculated from several records.

Each external view is defined by means of an ***external schema*** (also called ***subschema***), which consists basically of definitions of each of the various external record types in that external view. The external schema is written using the DDL portion of the user's data sub language. (That DDL is therefore sometimes referred to as an **external DDL**). Each user's schema gives a complete description of each type of external record that appears in that user's view. The schemas are compiled by the DBMS and stored in object form for use by the data dictionary in retrieving records if

the data dictionary is an active one. They should also be kept in source form as documentation.

At the external level, models or user-views are used by system analysts and users to elicit data requirements. They are used in training to explain to new employees how an information system is used. End users may interact with a database by using a query language or problem-oriented language.

The users of data are either programmers, transaction processing initiators (for example, airline ticket agents or bank tellers, Computer-Aided Design/Computer Aided Manufacturing (CAD/CAM) system users or ad-hoc query system users. Needs of these classes of user will span the entire range of possibilities of data models from tables for some clerks and managers to complex hierarchies or networks for the representation of parts and components in a CAM system. Though some commercial systems now have multiple interfaces, some problems still arise in general. The idea of a multiplicity of different model views is, however, now being realized.

To develop a user's view, the database designer first interviews the user and examines reports and transactions that he or she creates or receives. After the entire design of the database is planned, the DBA determines what data will be available to that user and what representation the user will see and writes an external schema for that user. Whenever possible, this external schema includes all the information the user has requested. Over time, the user's needs may change and modifications may be made to the view. Often, the new data required is already present in the database, and the user's external schema is rewritten to allow access to it.

External data models have two subcategories;-

- Logical external data models or views used to elicit and describe data requirements in a technology-independent manner.
- Sub-schema data models, which describe only the data required for a given data processing task which are defined using a technology dependent style (like a COBOL Data Division, FORTRAN dimension statement, DDL of a DBMS etc.)

The DDL used to define external views depends on the data model being used. Normally the data model used to define external schemas is the same as the one used in the conceptual schema (eg. both are network).

The external views for ORDER and the associated external schemas are shown in figure 4. These views are derived from the conceptual model by specifying relational commands that are applied to the relations CUSTOMER and ORDER.

There will be many distinct external views, each consisting of a more or less abstract representation of some portion of the total database.

In the relational model, *views* correspond to the ANSI/SPARC external level. Actually, most relational products on the market today muddy the external/conceptual distinction somewhat, because they allow users to operate directly on base tables as well as on views.

```
CREATE VIEW ORDER_VW
AS
SELECT ORDERID, CUSTID CUSTNUM,
(DELIVERYDATE – ORDERDATE)
DAYS_FOR_PROCESSING
FROM ORDER;
```

```
CREATE VIEW CUSTOMER_ORDER_VW
AS
SELECT ORDERID, CUSTNAME,
ORDERDATE, DELIVERYDATE
FROM CUSTOMER, ORDER
WHERE CUSTOMER.CUSTID =
ORDER.CUSTID; // using inner join
```

Fig 4: External Schema using DDL

INTERNAL LEVEL

Introduction

It is the database structural level defining physical view of database. The internal level covers the physical implementation of the database to achieve optimal run-time performance and storage space utilization. It covers the data structures and file organizations used to store data on storage devices. It interfaces with the OS access methods (file management techniques) for storing and retrieving data to place data on the storage devices, build the indexes, retrieve the data and so on.

This level is the responsibility of physical database designers/database administrator who decide which physical devices will contain the data, what access methods will be used to retrieve and update data and what measures will be taken to maintain or improve database performance. No user, as a user, is concerned with this view.

Internal View

The internal view is a low level representation of the entire database; it consists of many occurrences of each of many types of internal record (***Internal record*** is the ANSI/SPARC term for the construct that is also called ***Stored record***). The internal view is still at one remove from the physical level, since it does not deal in terms of physical records – also called blocks or pages – nor with any device-specific considerations such as cylinder or track sizes. In other words, the internal view effectively assumes an infinite linear address space; details of how that address space is mapped to physical storage are highly system-specific and are deliberately omitted from the general architecture. (Note: The block or page is the unit of I/O i.e. it is the amount of data transferred between secondary storage and main memory in a single I/O operation. Typical page sizes are 1K, 2K or 4K bytes (1K = 1024 bytes).

Internal Schema

The internal view is described by means of the internal schema. *The internal schema is used to comprehensively define the whole database using a technology-dependent style - that is models limited by the capabilities of the technology and that explicitly states how the technology will be used to manage data.* The DBMS chosen determines, to a large extent, what structures are available.

The internal schema is concerned with such things as

- Complex low-level data structures and file organizations used to store the data on physical storage devices.
- Indexing and Hashing schemes used
- Record descriptions for storage (with stored size for data items)
- The methods of representation of data fields
- Storage space allocation for data and indexes
- Record placement[1]
- Physical sequence of records
- Data compression and data encryption techniques

The internal schema defines the details of the data structures that are used by the DBMS to locate records and to establish associations between records. For example, inverted lists that are used to implement secondary keys are defined in the internal schema as are pointer fields used to link records.

The idea of an internal schema allows the DBA to decide what internal efficiencies and storage methods are appropriate for the organization and possibly also how they are mapped to the external schema.

The internal schema is written using another data definition language, the internal DDL. All DBMS vendors also support SQL commands to describe aspects of the physical schema but these commands are not part of the SQL-92 language standard.

Interface with Operating System

The DBMS works with the operating system access methods to lay out the data on the storage devices, build the indexes, and/or set the pointers that will be used for data retrieval. Therefore, there is actually a physical level below the one the DBMS is responsible for, one that is managed by the OS under the direction of the DBMS. The line between the DBMS responsibilities and the OS responsibilities are not clear cut and vary from system to system. Some DBMSs take advantage of many of the OS access method facilities, while others ignore all but the most basic I/O managers and create their own alternative file organizations. The physical level below the DBMS consists of items only the OS knows such as exactly how the sequence is

[1] The PCTFREE clause in Oracle SQL (e.g. PCTFREE 10): Amount of space reserved in each block of segment for future updates of existing rows.

implemented and whether the fields of internal records are stored as contiguous bytes in the disk.

An **internal record** is a single stored record. It is the unit that is passed up to the internal level. The **stored record interface** is the boundary between the physical level, for which the OS may be responsible, and the internal level, for which the DBMS is responsible. This interface is provided to the DBMS by the OS. In some cases where the DBMS performs some OS functions, the DBMS itself may create this interface. The physical level below this interface consists of items only the OS knows, such as exactly how the sequencing is implemented and whether the fields of internal records are actually stored as contiguous bytes on the disk. The OS creates the physical record interface, which is a boundary below which storage details such as exactly what portion of what track contains what data are hidden.

The relational model has nothing to say regarding the ANSI/SPARC internel level. In principle, the system is free to employ any storage structures it likes at the internel level, provided only that it can abstract from those storage structures and present the data at the conceptual level in pure tabular form.

In many relational products, user request, in the form of SQL statements, make no direct reference to access structures such as indexes. As a result, the DBA or DBMS can create and destroy such structures freely for performance and tuning reasons, without invalidating existing applications.

MAPPINGS

Mapping is the mechanism that provides correspondence between two levels in the three-schema architecture of a database system. It is the means of providing isolation between the different levels.

The DBMS is responsible for mapping between these three types of schema. It must also check the schemas for consistency; in other words, the DBMS must check that each external schema is derivable from the conceptual schema and it must use the information in the conceptual schema to map between each external schema and the internal schema. The data dictionary not only stores the complete external, conceptual and internal schemas, but it also stores the mappings between them. There are two levels of mapping in the architecture, one from the conceptual level to the internal level and one from the external level to the conceptual level.

The ***conceptual/internal mapping*** defines the correspondence between the conceptual objects and internal ones; it specifies how the conceptual records and fields are represented at the internal level. If the structure of the stored database is changed – i.e., if a change is made to the storage structure definition – then conceptual/internal mapping must be changed accordingly, so that the conceptual schema can remain invariant. In other words, the effects of such changes must be isolated below the conceptual level, in order that physical data independence might be preserved. It is the responsibility of the DBA to manage such changes, of course. The conceptual/internal mapping is the key to physical data independence.

The conceptual/internal mapping enables the DBMS to find the actual record or combination of records in physical storage that constitute a logical record in the conceptual schema, together with any constraints to be enforced on the operations for that logical record. It also allows any differences in entity names, attributes order, data types, and so on, to be resolved.

The ***external/conceptual mapping*** tells the DBMS which objects on the conceptual level correspond to which objects in a particular user's external view. There may be differences in record names, data item names, data item order, data types and so on. Moreover, several conceptual fields can be combined into a single (virtual) external field, and so on. If changes are made to either an external view or the conceptual model, the mappings must be changed. The external/conceptual mapping is the key to logical data independence.

Any number of external views can exist at the same time; any number of users can share a given external view; different external views can overlap.

Incidentally, most systems permit the definition of one external view to be expressed in terms of others (in effect, via an ***external/external mapping***), rather than always requiring an explicit definition of the mapping to the conceptual level – a useful feature if several external views are rather similar to one another. Relational systems in particular typically do provide such a capability.

The two-stage mapping in the ANSI/SPARC architecture may be less efficient but provides greater data independence. However, for more efficient mapping, the ANSI/SPARC model allows the direct mapping of external schemas on to the internal schemas, thus by passing the conceptual schema. This, of course, reduces

data independence, so that every time the internal schema changes, the external schema and any dependent application programs may also have to change.

ILLUSTRATING DIFFERENCES BETWEEN THE THREE LEVELS

An example of the different levels is shown in fig 5. Two different external views of staff details exist: one consisting of a staff number (*sNo*), first name (*fName*), last name (*lName*), age and salary; a second consisting of a staff number (staffNo), last name (lName), and the number of the branch the member of staff works at (branchNo). These external views are merged into one conceptual view. In this merging process, the major difference is that the *age* field has been changed into a date of birth field, *DOB*. The DBMS maintains the external/conceptual mapping; for example, it maps the *sNo* field of the first external view to the *staffNo* field of the conceptual record. The conceptual level is then mapped to the internal level, which contains a physical description of the structure for the conceptual record. At this level, we see a definition of the structure in a high-level language. The structure contains a pointer, next, which allows the list of staff records to be physically linked together to form a chain. Note that the order of fields at the internal level is different from that at the conceptual level Again, the DBMS maintains the conceptual/internal mapping.

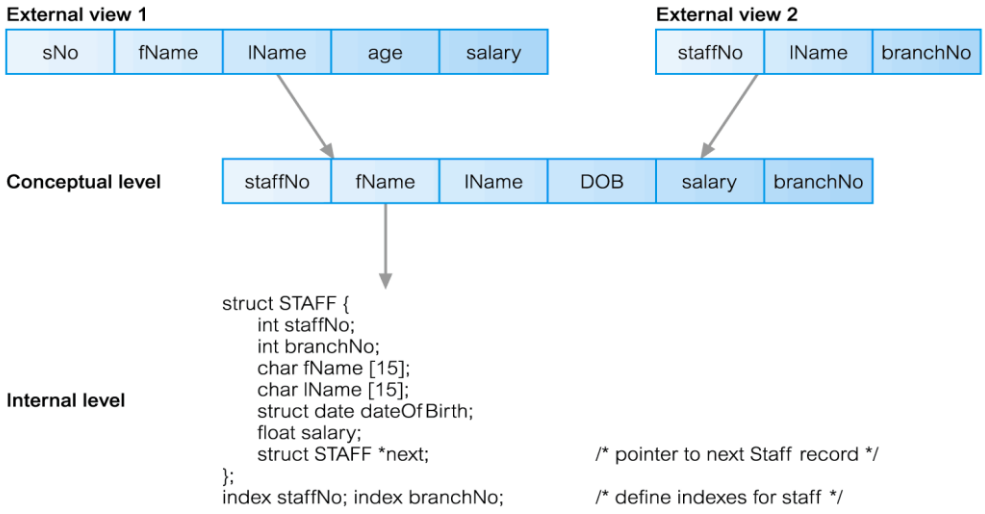

Fig 5: Difference between the three levels

CONCEPTUAL MODELING

From an examination of the three-level architecture, we see that the conceptual schema is the *heart* of the database. It supports all the external views and is, in turn, supported by the internal schema. However, the internal schema is merely the physical implementation of the conceptual schema. The conceptual schema should be a complete and accurate representation of the data requirements of the enterprise. If this is not the case, some information about the enterprise will be missing or incorrectly represented and we will have difficulty fully implementing one or more of the external views.

Conceptual modeling or conceptual database design is the process of constructing a model of the information use in an enterprise that is independent of implementation details such as the target DBMS, application programs, programming languages, or any other physical considerations. This model is called a conceptual data model. Conceptual models are also referred to as logical models by some authors. However, most writers make distinction between conceptual and logical data models. The conceptual model is independent of all implementation details where as logical model assumes the knowledge of the underlying data model of the target DBMS.

DATABASE DEVELOPMENT USING THREE-SCHEMA ARCHITECTURE

Together, conceptual, external and internal schemas form the three-schema architecture for databases. Note that even though figure x shows external schemas at the top, external schemas are not necessarily developed before the conceptual schema. In fact, we typically develop conceptual and external schemas iteratively. Often we develop a first cut at the conceptual schema based on the organization's *enterprise data model* and the general understanding of database requirements on a project. Then we develop external schemas (user views) for each transaction, report, screen display, and other system use. In most cases, an analysis of the external schemas will yield new attributes and possibly entities and relationships not shown in the conceptual schema. So, next we will augment the conceptual schema with these requirements identified from so called bottom-up sources, thus making the conceptual and external schemas consistent. When new user views are identified, this process of evolving both the conceptual and external schemas repeats.

When we are ready to begin developing the database and its associated application programs, we then write the specifications for the associated physical schema.

Besides the conceptual and external schemas, we additionally consider the software and hardware characteristics and users' database performance expectations while designing the physical database with its associated physical schema. It is possible that when we design the physical database we will encounter inconsistencies or other issues with the conceptual schema or user views, so it is possible to cycle back to these design steps.

When additional user requirements are identified later, the process begins again. Usually new requirements are batched and considered together in a new cycle of external and conceptual schema design, so that the database is not constantly changing. Periodic revisions of the database occur during the maintenance phase of systems development.

* * * * *