01. Database Basics

[PM Jat, DAIICT, Gandhinagar]

Contents

- What is a Database?
- How do we represent (Model) databases?
- Database [Instance] and Database Schema
- Database Abstraction and DBMS-based computing architecture
- Three Schema Architecture

What is a database?

In simple terms, a database is a collection of "Relevant" "Data"!

A database typically

- Records all facts [let us call them data] about some "application" context.
- Records necessary information of all events that occur as "business process"; for example, consider events in "online shopping" application that are "order is submitted"; "an order is shipped", "order is delivered". Each of these event shall add some data.
- Data are stored such that "data manipulation" operations can be executed efficiently. Adding of data, searching required data, and modifying exiting data, all are considered as "data manipulating" operations on databases.

To explain further, we elaborate terms "Relevant" and "Data" next.

Data

In databases, facts are conceived and represented as *entities* and their *attributes*.

For example, in an academic scenario: The **Student** and **Course** are the entities. Student ID, Name, DOB, Commutative Performance Index (CPI), etc. are the attributes of a student entity.

Here the term "data" refers to a value for an attribute of an entity. For example, "Amit Kumar" is the value for the name attribute of a student entity; "28-June-1986" value for the attribute "date-of-birth" of the student, and so forth. A set of values for various attributes describes an *entity*.

In databases, a datum is also said to be an *atomic* value. Data being atomic means "indecomposable". For example, consider said DOB above; can it be split into "28", "June", and "1986"? Individually they are not DOB, it is meaning value only when they are put together. SO we say that attribute DOB is "atomic". Similarly, the name value "Amit Kumar" cannot be split into its alphabets or even "Amit" and "Kumar" when everything is put together then only it is a value for this attribute.

Why does data need to be atomic?

Consider the table given below. For the human mind, it contains some data! But is it data for the machine too? To answer this, consider answering following queries from the given data table.

- What is CPI of the student having ID "200711005"
- List ID, and name of students from batch "2007" of progid 11

Can you a write program (and how simple that is) to answer said queries when the table is in one of the following formats?

- In doc format, or html/pdf format
- In a spreadsheet, like Excel format
- Stored as a table in a relational DBMS like PostgreSQL/MySQL/SQLite

It would be easier when stored in a DBMS!

Why is it so?

Two reasons!

First, Data are stored as "atomic" that is stored at smallest possible decomposition of data by retaining meaning of data.

Second, DBMS provides proper querying interface and simplifies the programming.

studentid	name	progid	batch	cpi
200711001	Charu Chawla	11	2007	6.12
200711002	Amit Khanna	11	2007	7.12
200711003	Kamla Kiran	11	2007	7.50
200711004	Raj Kumar	11	2007	4.00
200711005	Raj Tiwari	11	2007	5.56
200811001	Rama Kant	11	2008	8.12
200811002	Akshya Gupta	11	2008	9.22
200811003	Unnati Gupta	11	2008	5.52
200811004	Mridula Singh	11	2008	4.25
200811005	Amit Ajaad	11	2008	6.56

Figure 1: Sample data - Student Records

Coming back to the "Atomic" which main pint of discussion here. Here the main question is when data is "indecomposable"? We have various levels of composition of data in the example in discussion here. Table as a single unit, or table as collection of rows; where row is storage unit? Or, we further break each row as a set of *attribute-value* pairs. Now coming to attributes, same question also pops up, do we store *name* as single attribute or further decompose it further into first name, last-name, or so. ProgID and Batch both information are there in StudentID, still we store them separately? Why? Everything comes down to "atomic" attributes.

Let us conclude it with simply saying "break it to smallest possible level without loosing any meaning of the data".

Data are broken into pieces that meaningfully represent the facts. "Atomic" "Attributes" of the entity, and let the value of an attribute of an entity be referred to as "data". Then data are stored such that the machine is able to identify and locate it by its name.

This makes the data machine processible.

It is difficult (for machines) when you have data in text/html/doc/pdf.

When represented in Excel, a data item is locatable, but not by name but by cell location.

This is why the database approach advises us to store data in its smallest possible granularity, that is "atomic"!

Relevant Data

A database does not have a random collection of data. All data that a database store has to have "relevance".

Different people have different ways of defining "relevance". Book Elmasri/navathe uses the concept of "mini real world" or "universe of discourse" to define the relatedness of the data. The book Korth uses the notion of "enterprise" to define the "boundary" between what is relevant and what is not?

Basically, intuition here is, that we are able to draw some boundary to separate out "relevant data of the database" in the data of universe.

Let us use a very simple way to define relevance. Databases are built for some purpose. Let the data be called as "relevant" if data is required for meeting the objective of building the database.

For example, consider student database in an academic institution. Then we ask a question does "date of birth" of student is required for this database? Mostly answer is Yes – then we call it a relevant data? Does Father's name of student is required in a university database? Mostly yes. Do we require Mother Name? Probably Yes. Do we require sibling's name? Probably No.

Hereby we say that data is "relevant" if it is required in the database, otherwise not.

Representation of databases

We have certain data modeling techniques that are used for representing databases. Following are two most popular techniques.

- Entity-Relationship Models (Conceptual Model)
- Relational Model (Implementation Model)

Entity-Relationship Models:

- Database is seen as set of *entities* of different types and relationships (interactions) between them.
- ER model is "Conceptual Model", and primarily used for documentation purpose.

Example #1 (Entity-Relationship Models)

Consider a simple scenario of X Institute of Technology (XIT) where, let us say we have entities: Student, Program, Department. Students are related to students Program as they study in some program. Programs are offered by a Department, therefore Program and Department are related though this relationship.

In ER Model for this scenario, we have a set of Student, Program, and Department entity sets and their relationship sets. Where we describe an entity by values [data] for all of its attributes.

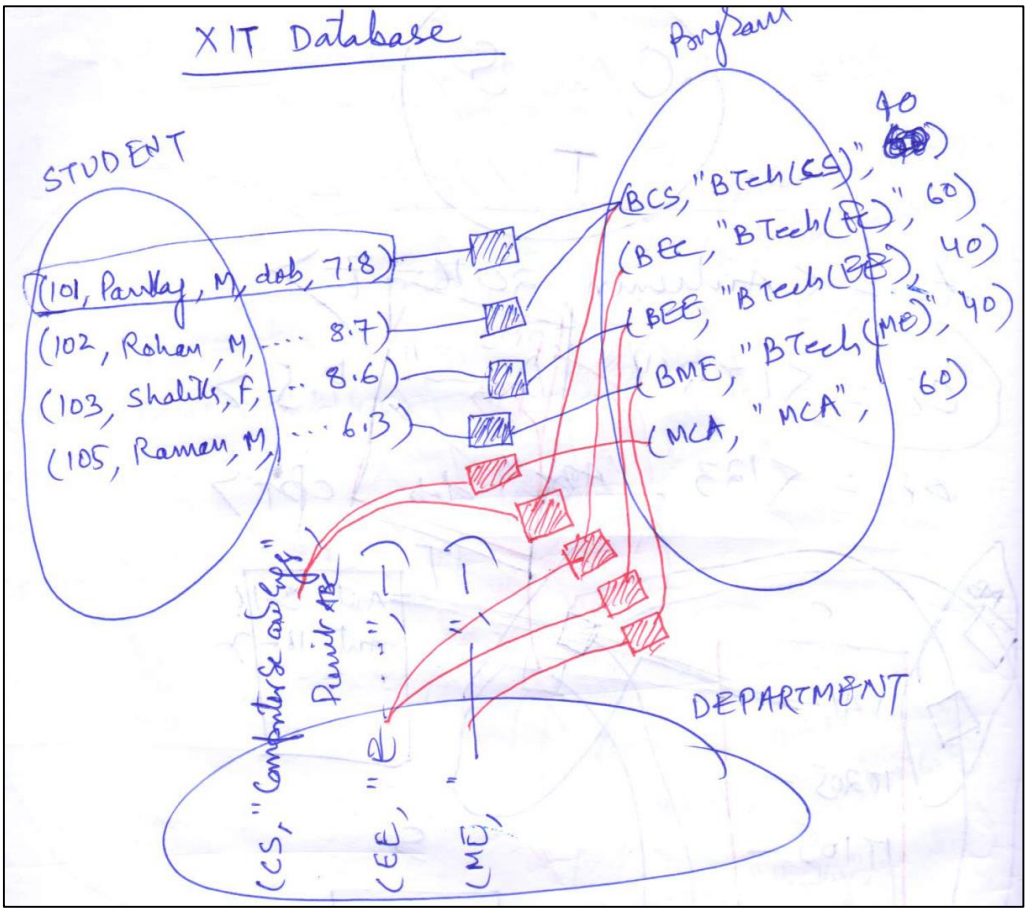


Figure 2: [XIT-ER] Database in terms of Entities and their interactions

XIT Database = {S, P, D, SP, PD}

Where, `S = {s1, s2, s3, …}` - set of students such that $s \in S$ is set of attribute value pairs, for example: `s1 = {ID=102, Name="Pankaj", dob="22-10-92", cpi=6.78}`

`P = {p1, p2, p3, …}` - set of programs

`D = {d1, d2, d3, …}` - set of departments

`SP` = set of events/instances of students enrolling in programs. A student can enroll only in one program and a program can have multiple students registered.

`PD` = set of events/instances of departments offering programs. A department can offer multiple program and a program is associated with only one program.

Example #2 (Entity-Relationship Models)

Figure below partially depicts a database scenario "academic records of a university"

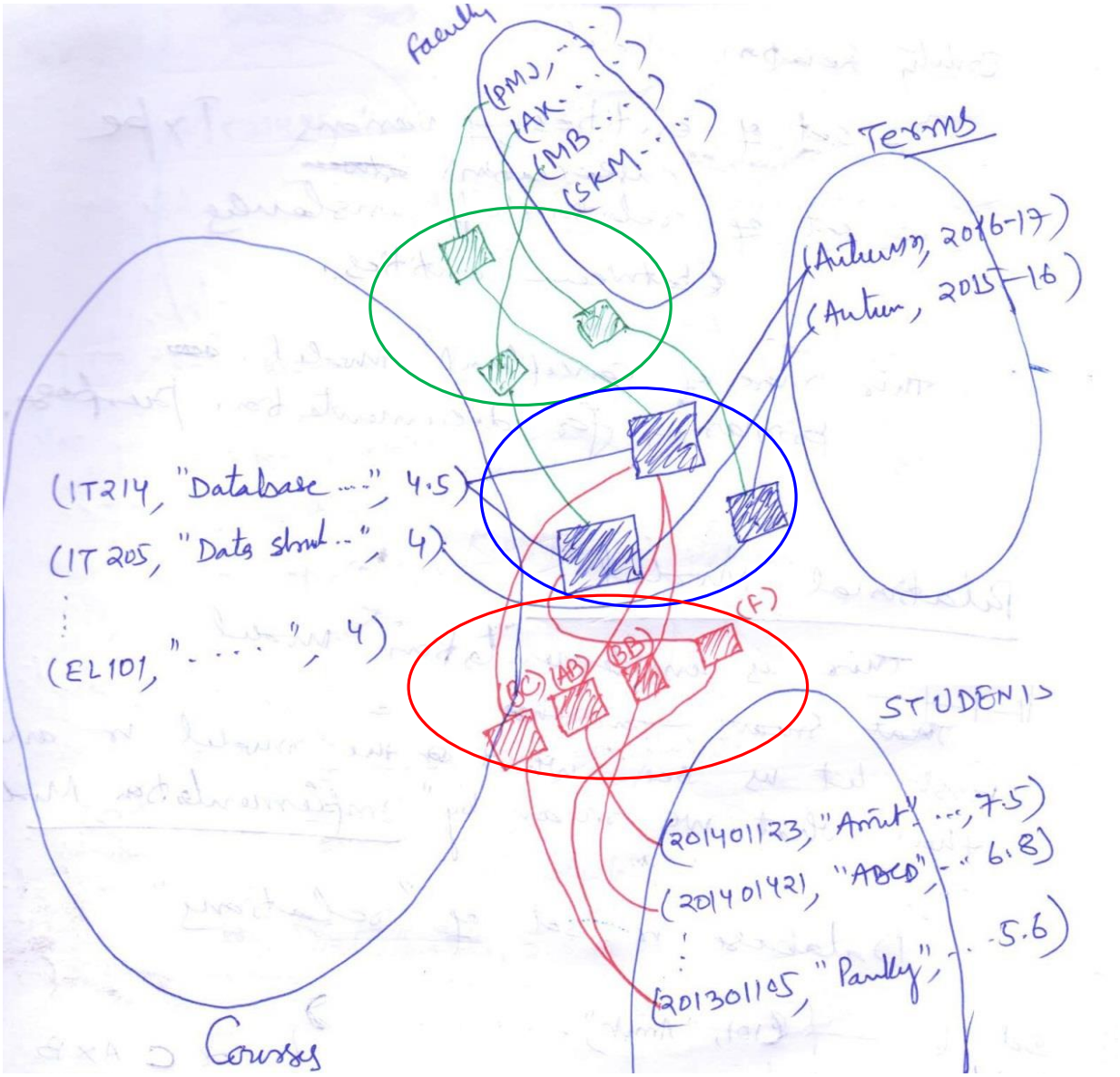


Figure 3: ACAD (partial) Database in terms of Entities and their interactions

`ACAD DB ={S, C, T, F, CT, CTF, CTS}`

Where

`S = {s1, s2, s3, …};` set of students

`C = {c1, c2, c3, …};` set of courses

`T = {<Autumn,2016>, <Winter,2017>, … };` set of terms

`F` = set of faculty

`CT` = Set of instances of Course offerings (set of tiny blue-boxes)

`CTF` = set of instances of "course offering" getting a faculty associated (set of tiny green-boxes)

`CTS` = set of instances of students registering in offerings. (set of tiny red-boxes)

Relational Model (Implementation Model)

- In this technique, Database is represented as a set of *relations*, also called as tables. In other term "database is a set of tables".
- Each relation/table is set of tuples, or rows.
- Relational model is de-facto standard for implementing enterprise databases.

Example (relational representation):

XIT Database – set of relations/tables

```
XIT-DB = {Student, Program, Department}
```

Where Student, Program, Department are three relations as shown below.

Student

StudentID	Name	ProgID	CPI
101	Rahul	BCS	7.5
102	Vikash	BIT	8.6
103	Shally	BEE	5.4
104	Alka	BIT	6.8
105	Ravi	BCS	6.5

Program

ProgID	ProgName	Intake	DID
BCS	BTech(CS)	40	CS
BIT	BTech(IT)	30	CS
BEE	BTech(EE)	40	EE
BME	BTech(ME)	40	ME

Department

DID	DName
CS	Computer Engineering
EE	Electrical Engineering
ME	Mechanical Engineering

Figure 4: XIT database in terms of Relations

Recap:

- What is database? Database is a collection of "relevant" "data"
- When Data is Data?
- Database Representation.
  (1) ER Model: Set of *entities* and their *interaction instances*!
  (2) Relational Mode: Set of relations

  And few examples

Database Instance, state, and Schema

The term "database" that we have been learning so far, refers "database instance". All sketches seen so far depict *database instance*.

A database instance is basically collection of all "values" in the database. At point of time instance represents state of the database and referred as *database state*. In any real system, databases keep updating; and continuously changing its state.

We have another concept "database schema".

Database Schema-

While database instance holds data; **database schema** describes "structures of database". Database structure, implicitly, also includes a description of *database constraints*.

In other words, database schema contains description of database structure and its constraints.

Database Structure

Database describes

- What entities? Or what relations if represented in relational model.
- What "attribute values" for each entity (or relations)

Database Constraints

Defines rules that a data in database should comply; mostly includes:

- What is "domain" for each attribute of an entity (or relation)?
- What are the key attribute(s) for entities (or relations)?
- What interaction it has with other entities? is that interaction mandatory?
- What is the cardinality of interaction? That is, to how many entities of another type, an entity can interact with and vice-a-versa.

Below is a depiction of XIT database schema in "entity-relationship representation" (ER Schema), and relational representation (Relational Schema)

ER Schema	Relational Schema

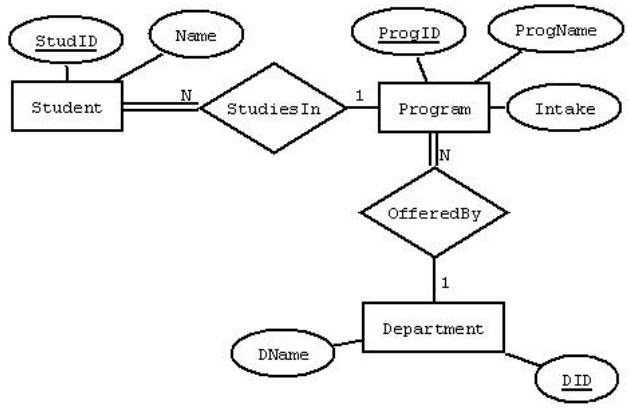
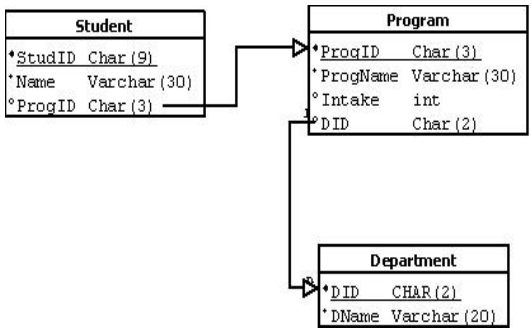

	

Figure 5: Schema of XIT database in ER Model	Figure 6: Schema of XIT database in Relational Model

Also below is, XIT schema in SQL-DDL

```
CREATE TABLE department (
      DID CHAR(2) PRIMARY KEY,
      DNAME VARCHAR(30) NOT NULL
);
CREATE TABLE program (
      PID CHAR(3),
      PNAME VARCHAR(20) NOT NULL,
      INTAKE SMALLINT,
      DID CHAR(2) REFERENCES department(did)
          ON DELETE SET NULL ON UPDATE CASCADE,
      PRIMARY KEY (PID)
);
CREATE TABLE student (
      StudID CHAR(3),
      Name VARCHAR(20) NOT NULL,
      ProgID CHAR(3) REFERENCES program(pid)
          ON DELETE CASCADE ON UPDATE CASCADE,
      cpi decimal(4,2)
);
```

Figure 7: XIT schema in SQL-DDL

Here we have three different description of database schema – ER, Relational, and SQL-DDL. Each description serves some specific purpose.

- ER description is often used for documentation purpose.
- Relation description is used for documenting "implementation structure of database".
- Schema in SQL DDL script, as a program, becomes input to a database management system to create **empty database instance** on the computer system as per specified schema in DDL form.

Database Constraints

Database constrains are basically "data existential rules". "Rules" that must hold true on data (values) in a database.

For example, in ACAD database

- StudentID is key attribute of student entity; that means no two student entities can have same value for this attribute.
- A course offering necessarily need to have an instructor associated with; and exactly one instructor.
- An elective can be taken only from respective domain, and so forth

Any data that violate such rules cannot exist in database; if do, then database is said to be in invalid state; such a state is called as "**inconsistent state**" of database.

The interpretation of "Inconsistent", here is database is not consistent with its constraints (rules)

Constraints are part of "database description", referred as *database schema*, and any valid state of the database should satisfy these rules.

Meta-Data: Database Schema as Data

One of the characteristics of the database is self-describing. The dictionary meaning of the term metadata is "data that provides information about some other data". In databases, metadata refers to the "data about the database", that is schema information of the database "**as data**". DBMS stores schema information of every database in its "catalog".

Suppose XIT database is already created on a PostgreSQL server. We can query the server to get the schema of XIT. Here is a snapshot of queried schema information from the catalog. The query used to fetch this information is given in footnote [1]

table_name character varying	column_name character varying	data_type character varying
department	did	character
department	dname	character varying
student	studid	character
student	name	character varying
student	progid	character
student	cpi	numeric
program	pid	character
program	pname	character varying
program	intake	smallint
program	did	character

Figure 8: Database Metadata

Operations on Databases

The following are the main operations on the database (also referred to as database manipulation operations). We have two types of database operations.

Update operations:

These are operations that add/modify/delete data from the database. Operations that change data of databases. There are three type of update operations: INSERT, MODIFY, DELETE.

- **INSERT**: insert operation adds more facts that is add more entities and their interaction. Example: Add a new student (id=234, Name='Aman', ..) in program BCS
- **MODIFY**: modify operation modifies exiting entities and their facts. Example: Update CPI of a student (id = 102) to 7.8
- **DELETE**: The operation deletes existing entities. For example, remove a student with ID=234 or delete the program 'BIT'

It is the business events that trigger the database update.

For example, the following events will be updating the corresponding sets in "ACAD" database –

- Add a course - adds an element in set C
- Add new Term - adds an element in set T
- Offer a course in a term - adds an element in set CT
- A faculty is assigned to a course - adds an element in set CTF
- A student registers in a course – adds an element in set CTS
- Faculty uploads course grades – modifies attribute values of elements in set CTS (for all students registered in given course in given term)
- Result processing – computes and updates SPI and CPI of each student for current semester!

[1] SELECT table_name, column_name, data_type FROM information_schema.columns
WHERE table_name in (select tablename from pg_tables where schemaname='xit') and table_schema = 'xit';

Querying operations:

The operations that answer queries from the database are called querying operations. Below are some examples:

- List (ID, Name) of BCS students having CPI > 7.0
- Give me a program-wise student count for all programs
- How many students are studying in the CS department?

Challenges in performing database operation

A very important work that we have in database management is "efficient execution of data manipulation operations". There are two following tasks that almost every database application requires doing, irrespective of what the data entities are:

(1) How do we "organize data records" on disk that enable us efficient execution of data manipulation operations? Here, let a data record be defined as a group of data, typically representing an entity. Common File Organizations are "Un-Ordered", "Ordered", "Binary Search Trees"

(2) What algorithms do we use for performing data manipulation (Insert, Update, Delete, and Search) operations? Algorithms are normally designed for "access paths", and require appropriate access paths to be available. Common access paths are "sequential", "binary search", "binary tree search", "hashing", etc.

Issues that we have while performing these tasks on stored data files are outlined as follows:

(1) Maintaining an efficiently searchable data file is a complex task. Data are stored in disk files and stored such that data updates and searching are efficiently done. Suppose, the B+ tree is an efficient approach for all this, then implementing this on secondary storage is complex. We would want such complexities to be transparent from applications.

(2) Repeatable Data manipulation functionality. We perform the following operations for almost every database application. We would not want this to be part of every application, and it is desirable that all this functionality is externally provided and remains reusable for applications.

(3) Independence of "Application Code" and "File Organization. The application's data access code is dependent on the underlying file organization. As a result, we cannot independently improve upon file organization. Requires changing the application code as well.

(4) Concurrency Issues. Issues related to concurrent access by multiple users

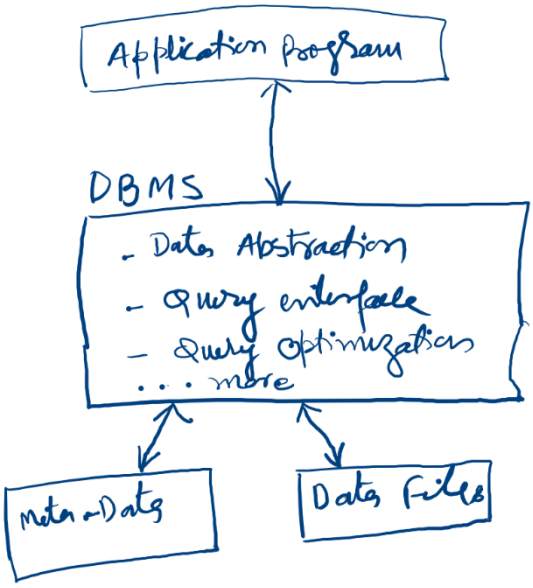
(5) Others: authorized access, dealing with system failures, or so

DBMS based Computing

A DBMS-based Computing Architecture was suggested to deal with said issues; figure here depicts the approach.

In this approach, DBMS sits in between of "application programs" and "Data Storage" and provides the following functions

1. First and most important is "**Logical Data Abstraction**" over the physical storage of data. This means DBMS allows performing data manipulation operations in some "logical manner". For example, databases are manipulated as tables while hiding internal representations of data on disks.
2. Provides a simple data manipulation interface – typically a query language like SQL.
3. Others being
    a. Query Optimization
    b. Ensures "database integrity". DBMS would reject a manipulation operation on a database that violates database constraints (data existential rules).
    c. Transaction Management: safe shared concurrent access by multiple users and recovers from system failures.
    d. Database Security: Authorization based accesses



When user submits database operation to DBMS using interface language like SQL; it in turn translates the request into file manipulation system calls or instructions, after checking syntax etc.

DBMS stores data on disk files; also keeps schema information of every database in its "dictionary" as meta-data.

DBMS maintains sophisticated file organization and indexes for accessing data in data file. Also DBMS implements complex set of algorithms to perform various user operations.

DBMS Definition:

Book Elmasri/Navathe gives following definition of DBMS-

"DBMS is general purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing the databases among various users and applications"

Data Abstraction

What is Data Abstraction in general? Consider example below, hopefully illustrates the notion? Observe how floating points are internally represented, and relevant operations are actually performed. User is transparent to underlying binary representation, and procedure of performing floating point operations.

There are two kind of transparency we have in "data abstraction"

- Representation Transparency (how data re represented)
- Operational Transparency (how operations are performed - procedure)

Programming languages do provide such *typed abstraction* over binary representation and manipulation of data.

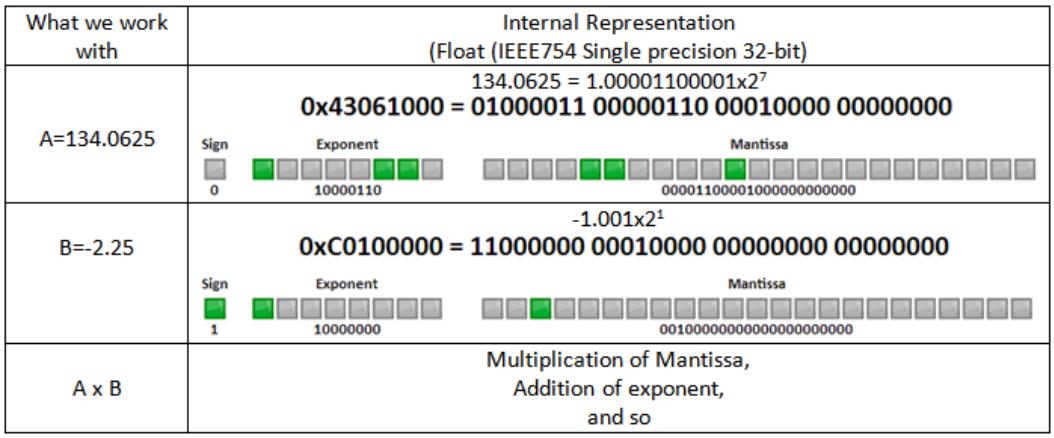


Figure 9: Programing Languages provide typed abstraction over binary representation of floating point numbers

Data Abstraction by DBMS

The same notion of data abstraction is applied to relations. DBMS provides -

- Representational Transparency: allows representing the database in some logical view while hiding how data are stored in disks on the file system.
- Operational Transparency: enables performing operations on the database without letting us delve into underlying complex algorithms.

For example, Relational is a popular database representation and manipulation model. RDBMS is DBMS based on the Relational model and provides representational and operational transparencies.

- Defining and manipulating databases as relations or tables while data are stored on disks.
- Perform various manipulation operations on relations in a logical manner (rather than how they are performed by RDBMS). DBMS uses sophisticated algorithms for performing database manipulation operations (add/modify/delete, and query/search)

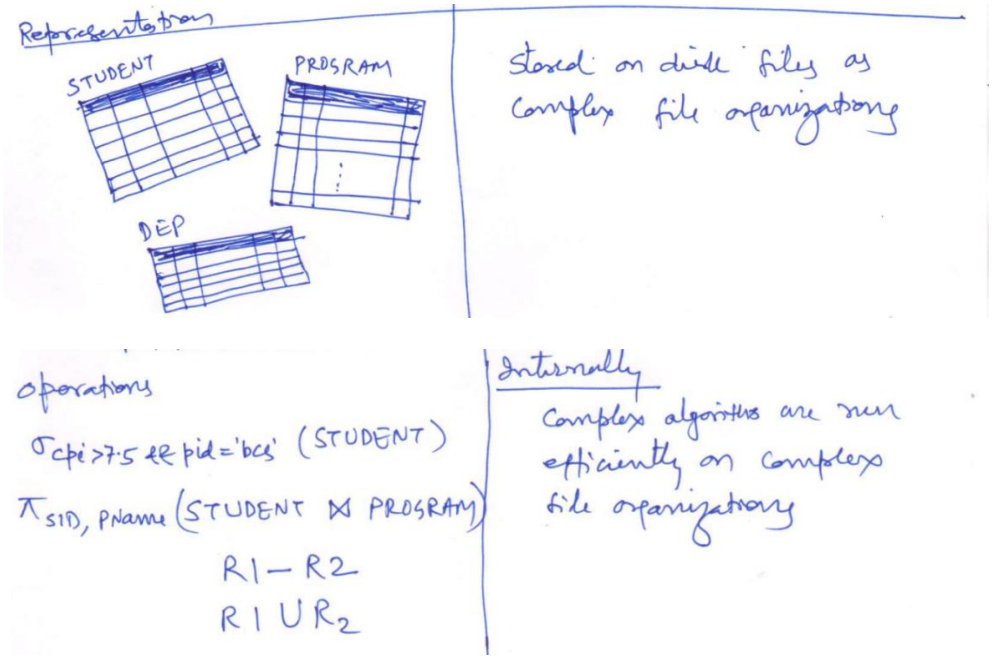


Figure 10: Relational abstraction

RDBMS provides SQL support for manipulating databases. Database is manipulated as tables, rather than data structure on disk files.

This way RDBMS hides the complexity of performing operations. Below are some SQL statements for certain database operations-

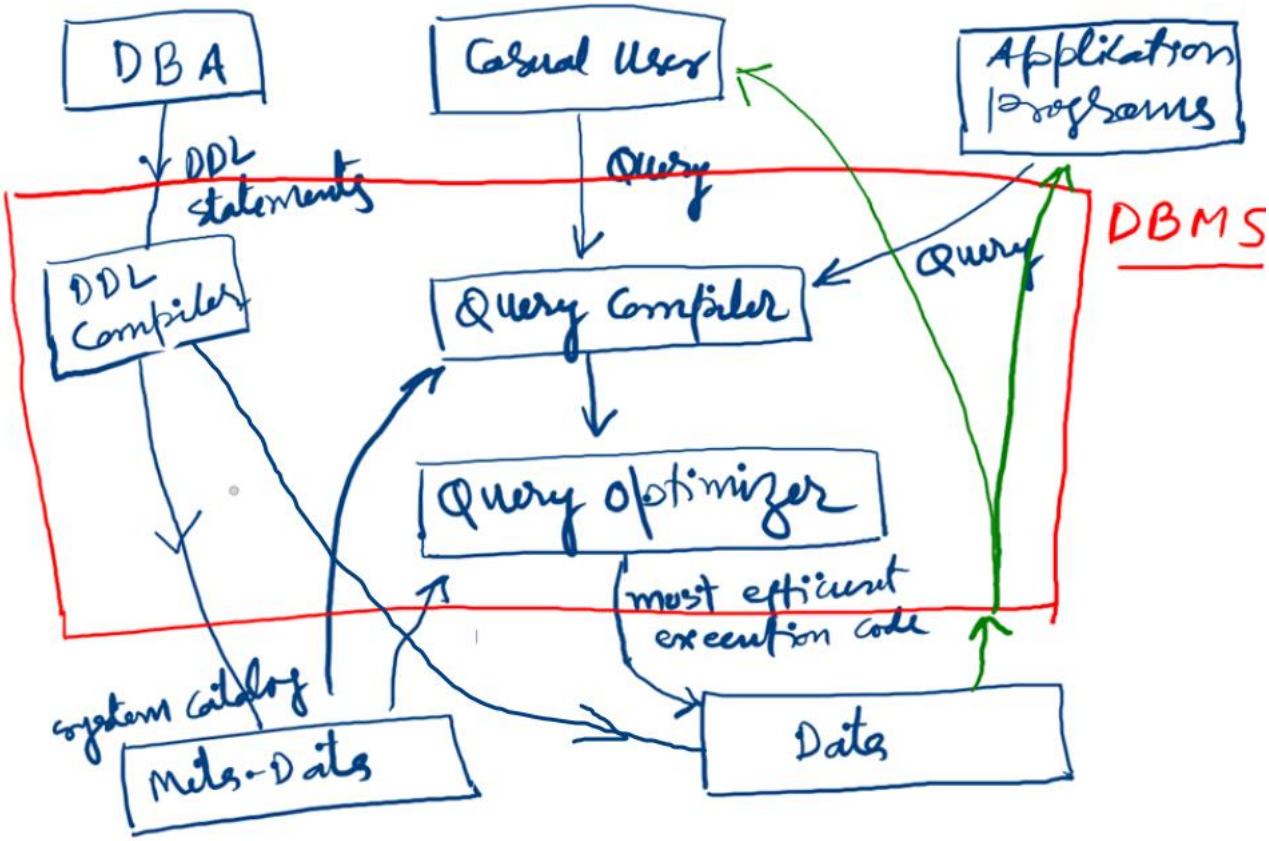
```
INSERT INTO STUDENT VALUE('201001123', 'Ankit', …);

UPDATE REGISTERS SET GRADE = 'AB' WHERE STUDENT_ID = '201001123'
          AND COURSE_NO = 'IT321' AND ACAD_YEAR = 2012;

SELECT * FROM STUDENTS WHERE CPI >=8.0 AND BATCH = 2009;
```

Actual algorithms that perform these operations are quiet complex. They work on disk files corresponding to concerned relations.

Working of a DBMS



The downside of using DBMS

- Requires more resources.
- Due to a translation and computing layer in between, performance goes down
- Deployment of applications becomes complex.
- Due to these reasons, many popular proprietary applications do not (did not) use DBMSes; reasons could be: performance, cost, proprietary, compact, etc.
- About a decade back mobile or other devices could not afford to use DBMS due to limited resources. Today you have SQLite and similar products, though!

Three Schema Architecture

American National Standards Institute (ANSI) and Standards Planning and Requirements Committee (SPARC) identify three levels of defining database schema (structures).

ANSI advises implementers of DBMS to follow three schema architectures.

Why?

Motivation of Three-Schema Architecture:

Recall that abstraction gives two very important benefits

1. Simplicity to the User
    a. Representational Transparency
    b. Operational Transparency
2. Independence to the Implementation

The idea of "Three-Schema Architecture" is to provide two-level "abstractions" for the different sets of "objectives".

Implementation Level

The objective of this level is to implement all DBMS functionality while hiding all physical level details: that is "representation" and "operation" transparency.

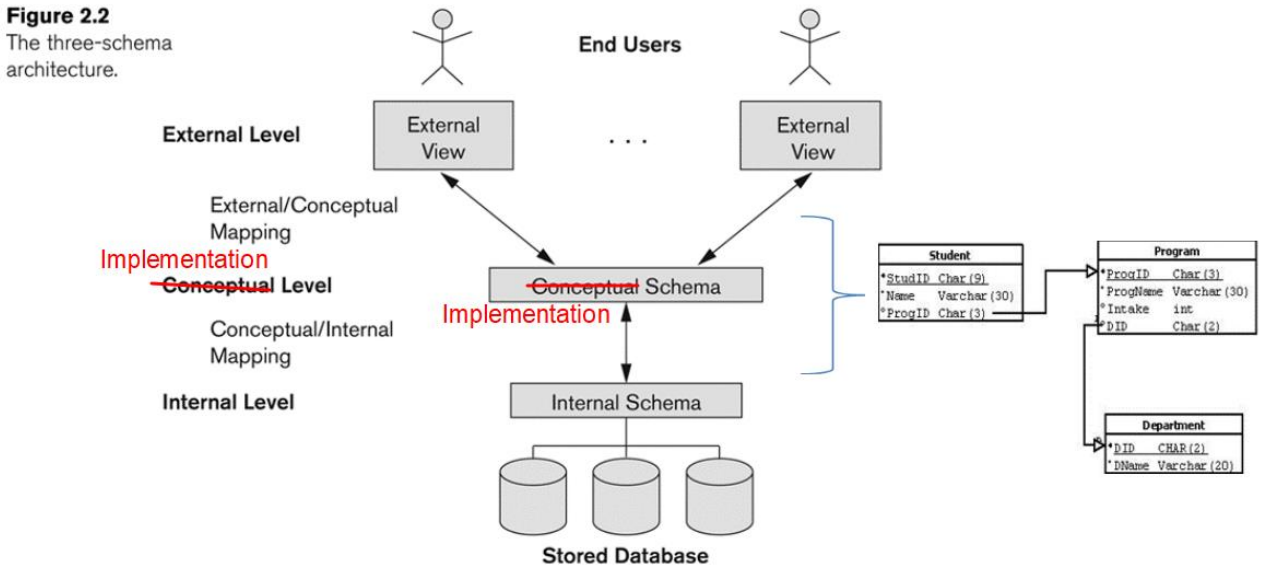
User Level / Application level

Application-specific abstraction over the implementation level. For example, an Object Oriented View over relational. The Objective of this level is to make the abstraction further specific to the user or application environment.

Recall that abstraction enables "independence" at a lower level without affecting the interface (abstraction). For example, lower-level data/file structures can be modified (for improvement) without affecting the high-level interface (abstraction).

Physical level, algorithms used for performing data operations are dependent on file organization. To improve the efficiency of execution of data operations, it may be required to change the file organization and access paths. Data independence enables that.

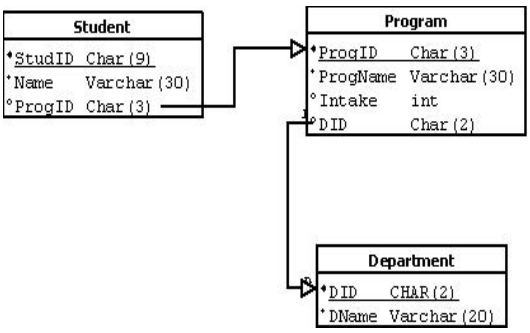
The diagram below from book elmasri/navathe depicts the architecture.



We have three layers of schema.

Middle Layer – Implementation Schema

Here is an example of a database that is defined in a logical manner, i.e. Relational. Here the database instance is represented by a set of "Tables"; where each Table has its Schema (and constraints).



Lowest Level - Physical/Internal Schema

- Data are stored physically stored on disk files.
- Database operations are executed by scanning disk files.
- Data are stored in Complex File Structures and often searches are enhanced by providing access paths like B+-Tree, hashing, or so.
- Internal schema primarily means "File organization" and "Data Access paths". File organization means the layout of data records on a disk file. File organization also includes ordering of data records in some attribute's order while storing.
- Indexes are used for fast lookup: B+ tree and Hashing-based indexes are the most popular indexing techniques.

External Schema or User Schema

- Object here is one higher level abstraction over the logical one.
- Not implemented by any relational system, so far.
- Normally not found in most have user level database Different users might see different subsets of the database, and see a schema defined in an independent technique.
- Few examples:
    - One such example is an external user might see the relational view of some subset of the legacy system
    - A user might see an object view of the database while having a relational implementation
- Other uses of external schema could be hiding actual database schema.

How independence is accomplished?

Often database operations are initiated by a user at the external schema level; whereas actual execution happens at the internal schema level (data are stored at the physical level)

A data reference at external schema should map to data at the physical level. Therefore, DBMS maintains "mappings" from higher level schema to lower level schema, and from lower level to the next level up to the data on the disk.

This mapping helps in accomplishing "data independence".

If any change in schema at a lower level happens, appropriate changes are made in mapping!

The downside of three schema architecture?

Reaching a data item referred by the user at an external level to corresponding data at the physical level requires two-stage resolutions through mappings.

This may become too much computational overhead and takes in executing a database operation (query).

Also, there would be some data representational transformations happening for the appropriate schema model at the respective level.

Due to these reasons, most DBMSs, RDBMS at least, do not implement the full three-schema architecture and typically do not separate schema at the external level.

You can emulate though by creating wrappers on top of it. For instance, there are lots of Object Relational Mapping tools used in the industry, that act as bridges between relations and *objects*.

References

[1] Elmasri, R., et al. *Fundamentals of Database Systems*. Addison-Wesley, 2013.
[2] Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. *Database system concepts*. McGraw-Hill, 2011.