Database System Concepts

a- Schema and Instance

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 infrequently is change**. Figure 2 shows a schema diagram for the database shown in Figure 1; the diagram displays the structure of each record type but not the actual instances of records.

The actual data in a database may change quite frequently. For example, the database shown in Figure 1 changes every time we add a new student or delete student. The data in the database at a particular moment in time is called **instance** or **snapshot.**

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

Figure (1): A database that stores student and course information.

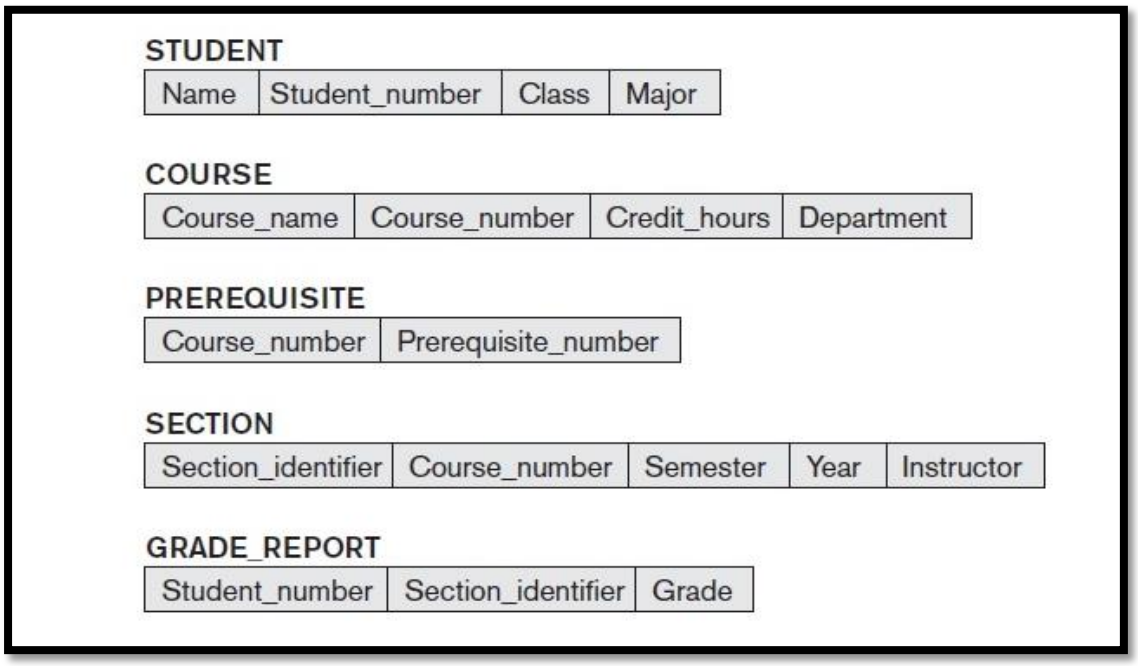

Figure (2): Schema diagram for the database in figure (1).

b- Data abstraction

One fundamental characteristic of the database approach is that it provides users with an **abstract view of the data**. Data abstraction generally refers to the hiding of details of data organization and storage. The complexity is hidden from users through several levels abstraction in order to simplify their interaction with the system.

Three-Schema Architecture

The goal of the three-schema architecture is to separate the user applications from the physical database. Figure (3) shows the three different schemas used in DBMS, seen from different levels of abstraction In this architecture, schemas can be defined at the following three levels:

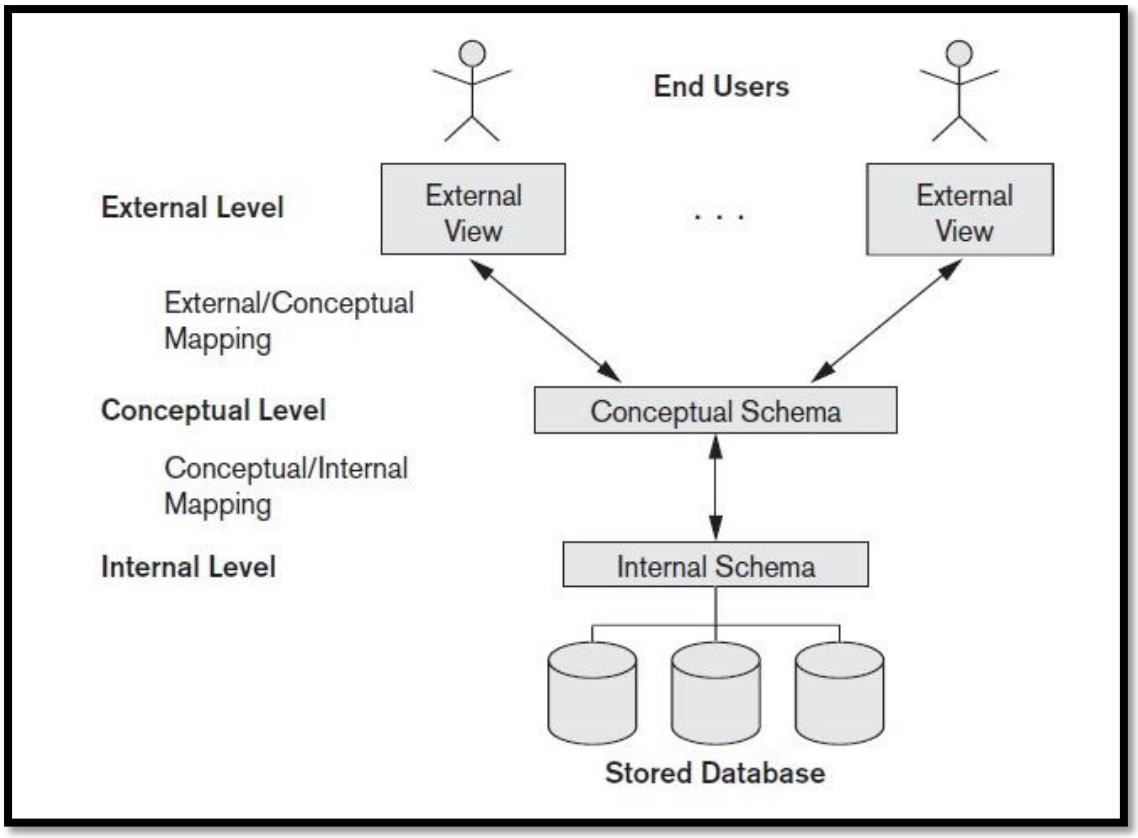

Figure (3): The three-schema architecture.

1. **Internal or Physical schema**, deals with the description of how raw data items are stored in the physical storage (Hard Disk, CD etc.). It also describes the data type of these data items, the size of the items in the storage media, the location (physical address) of the items in the storage device.

2. **Conceptual or Logical Schema**, 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. This means we want to know the information about the attributes of each table, the common attributes in different tables that help them to be combined, what kind of data can be input into these attributes, and so on. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This *implementation conceptual*

schema is often based on a *conceptual schema design* in a high-level data model.

3. **The external or view level** This is targeted for the end users. Now, an end user does not need to know everything about the structure of the entire database, 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. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

In a DBMS based on the three-schema architecture, each user group refers to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests and results between levels are called **mappings**.

c- Data Independence

The three-schema architecture can be used to further explain the concept of data independence, which can be defined as the ability 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: The ability to change the logical schema, without changing the external schema or user view, is called logical data independence. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item). For example, the addition or removal of new entities, attributes or relationships to this conceptual schema should be possible without having to change existing external schemas or rewrite existing application programs. In other words, changes to the logical schema should not affect

the function of the application. Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence.

Physical data independence: The ability to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update.

Whenever we have a multiple-level DBMS, its catalog must be expanded to include information on how to map requests and data among the various levels. The DBMS uses additional software to accomplish these mappings by referring to the mapping information in the catalog. Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the *mapping* between the two levels is changed. Hence, application programs referring to the higher-level schema need not be changed.

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....	....	.....
....	....	.....
....	....	.....
Prerequisite_number	XXXXNNNN	PREREQUISITE

Figure 1.3
An example of a database catalog for the database in Figure 1.2.

Data Item Name	Starting Position in Record	Length in Characters (bytes)
Name	1	30
Student_number	31	4
Class	35	1
Major	36	4

Figure 1.4
Internal storage format for a STUDENT record, based on the database catalog in Figure 1.3.

(a)

TRANSCRIPT

Student_name	Student_transcript				
	Course_number	Grade	Semester	Year	Section_id
Smith	CS1310	C	Fall	08	119
	MATH2410	B	Fall	08	112
Brown	MATH2410	A	Fall	07	85
	CS1310	A	Fall	07	92
	CS3320	B	Spring	08	102
	CS3380	A	Fall	08	135

(b)

COURSE_PREREQUISITES

Course_name	Course_number	Prerequisites
Database	CS3380	CS3320
		MATH2410
Data Structures	CS3320	CS1310

Figure 1.5
Two views derived from the database in Figure 1.2. (a) The TRANSCRIPT view. (b) The COURSE_PREREQUISITES view.

د يكون مهتمًا فقط بالوصول إلى نص كل منها وطباعته

طالب علم؛ يظهر عرض هذا المستخدم في الشكل ١.٥ (أ). مستخدم ثان مهتم

فقط في التحقق من أن الطلاب قد أخذوا جميع المتطلبات الأساسية لكل دورة

التي قاموا بالتسجيل من أجلها ، قد تتطلب طريقة العرض الموضحة في الشكل ١.٥ (ب).

لاحظ أن المخططات الثلاثة هي فقط أوصاف البيانات ؛ البيانات المخزنة التي

موجود بالفعل على المستوى المادي فقط. في DBMS على أساس المخططات الثلاثة

الهندسة المعمارية ، تشير كل مجموعة مستخدمين إلى مخططها الخارجي الخاص. ومن ثم ، فإن نظام إدارة قواعد البيانات

يجب أن يحول طلبًا محددًا في مخطط خارجي إلى طلب مقابل

المخطط المفاهيمي ، ثم في طلب على المخطط الداخلي للمعالجة

عبر قاعدة البيانات المخزنة. إذا كان الطلب عبارة عن استرجاع قاعدة بيانات ، يتم استخراج البيانات

من قاعدة البيانات المخزنة يجب إعادة تنسيقها لتتناسب مع العرض الخارجي للمستخدم. ال

تسمى عمليات تحويل الطلبات والنتائج بين المستويات بالتعيينات.

ينظر إليها من مستويات مختلفة من التجريد

توي المستوى الداخلي على مخطط داخلي يصف التخزين المادي

هيكل قاعدة البيانات. يستخدم المخطط الداخلي نموذج بيانات مادي

ويصف التفاصيل الكاملة لتخزين البيانات ومسارات الوصول لـ

قاعدة البيانات.

يحتوي المستوى المفاهيمي على مخطط مفاهيمي يصف الهيكل

من قاعدة البيانات بأكملها لمجتمع المستخدمين. المخطط المفاهيمي

يخفي تفاصيل هياكل التخزين المادية ويركز على الوصف

الكيانات وأنواع البيانات والعلاقات وعمليات المستخدم والقيود.

عادة ، يتم استخدام نموذج البيانات التمثيلية لوصف المفهوم

المخطط عند تنفيذ نظام قاعدة البيانات. هذا التنفيذ المفاهيمي

غالبًا ما يعتمد المخطط على تصميم مخطط مفاهيمي بمستوى عالٍ

نموذج البيانات.

يتضمن المستوى الخارجي أو مستوى العرض عددًا من المخططات الخارجية أو المستخدمين

الآراء. يصف كل مخطط خارجي جزءًا معينًا من قاعدة البيانات
تهتم مجموعة المستخدمين وتخفي بقية قاعدة البيانات عن ذلك
مجموعة المستخدمين. كما في المستوى السابق ، يتم تنفيذ كل مخطط خارجي بشكل نموذجي
باستخدام نموذج بيانات تمثيلي ، ربما يعتمد على نموذج خارجي
تصميم مخطط في نموذج بيانات عالي المستوى
ي القدرة على تغيير المخطط المفاهيمي
دون الحاجة إلى تغيير المخططات الخارجية أو برامج التطبيق. نحن
قد يغير المخطط المفاهيمي لتوسيع قاعدة البيانات (عن طريق إضافة ملف
نوع السجل أو عنصر البيانات) لتغيير القيود أو لتقليل قاعدة البيانات
)عن طريق إزالة نوع السجل أو عنصر البيانات). في الحالة الأخيرة ، المخططات الخارجية
التي تشير فقط إلى البيانات المتبقية لا ينبغي أن تتأثر. على سبيل المثال ، ملف
يجب ألا يتأثر المخطط الخارجي للشكل ١.٥ (أ) بتغيير
ملف) GRADE_REPORTأو نوع السجل) الموضح في الشكل ١.٢ في الملف
هو مبين في الشكل ١.٦ (أ). فقط تعريف العرض والتعيينات بحاجة إلى
يمكن تغييرها في DBMS الذي يدعم استقلالية البيانات المنطقية. بعد
المخطط المفاهيمي يخضع لإعادة تنظيم منطقية ، برامج التطبيق
يجب أن تعمل تلك الإشارة إلى بنيات المخطط الخارجية كما كانت من قبل. يمكن تطبيق
التغييرات على القيود على المخطط المفاهيمي بدون
تؤثر على المخططات الخارجية أو برامج التطبيق.
استقلالية البيانات المادية هي القدرة على تغيير المخطط الداخلي
دون الحاجة إلى تغيير المخطط المفاهيمي. ومن ثم الخارجية
المخططات لا تحتاج إلى تغيير كذلك. قد تكون التغييرات على المخطط الداخلي
مطلوب لأنه تمت إعادة تنظيم بعض الملفات المادية — على سبيل المثال ، من خلال إنشاء
هياكل وصول إضافية - لتحسين أداء الاسترجاع أو
تحديث. إذا بقيت نفس البيانات السابقة في قاعدة البيانات ، فلا ينبغي لنا ذلك
يجب أن تغير المخطط المفاهيمي.
عندما يكون لدينا نظام DBMS متعدد المستويات ، يجب توسيع الكتالوج الخاص به ليشمل
معلومات حول كيفية تعيين الطلبات والبيانات بين المستويات المختلفة. نظم إدارة قواعد البيانات

يستخدم برامج إضافية لإنجاز هذه التعيينات بالإشارة إلى التعيين
المعلومات في الكتالوج. يحدث استقلالية البيانات لأنه عندما يكون المخطط هو
تم تغييره على مستوى ما ، يظل المخطط في المستوى الأعلى التالي دون تغيير ؛ فقط
يتم تغيير التعيين بين المستويين. ومن ثم ، تشير برامج التطبيق
..إلى مخطط المستوى الأعلى لا يلزم تغييره