

CENG315
Information Managment Lecture Notes

Ege Özkan

October 15, 2020

Chapter 1

Introduction - October 15, 2020

1.1 Databases and Database Systems

Databases hold data. Database systems are software systems that manages the records in a database. There are five fundamental requirements for a database system.

- Database systems must be persistent, data must be storable and remain for the future.
- Databases must be able to handle getting large.
- Databases should be sharable, multiple users should be able to reach it at the same time.
- Databases must be kept accurate.
- Databases must be usable.

1.1.1 Record Storage

Databases can be made persistent in different ways.

Storing database records in text files

- Simplest approach.
- One file per record type.

- Each record could be a line of text, with its values separated by tabs.

1	joe	2020
2	amy	2013
3	lee	2000

Its advantages are the database system has to do very little, and a user could easily examine and modify the files with a text, but it is slow. (?)

1.1.2 Data Models and Schema

Data models are different ways to express connections between records while Schemas are the implementations of these methods for a specific database.

File-system v. Relational

In the file system model, each record type has a file, with one record per line, programs that read and write to the file is responsible for understanding this. In the relational data model, each record type has its **table** and each record has **fields** for each value. User access to the database happens via this record and field model and records that fit certain conditions can be queried.

These models are at different levels of abstraction, relational model is a **conceptual**

model, since there is no need to know **how** schemas are specified and implemented, the conceptual schema describes what the data *is*. Whereas the file-system is called a **physical model**, physical schemas say how the data is *implemented*.

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptID)
SECTION(SectId, CourseId, Prof, Year)
DEPARTMENT(DId, Name)
```

Figure 1.1: An example schema

Physical Data Independence

A conceptual schema is certainly nicer to use than a physical scheme. Operations on a conceptual schema is implemented by the database schema. Database system has a **database catalog** that contains descriptions of the physical and conceptual schemas. Given an SQL query, the database system translates the conceptual abstraction to the physical one and interact with it on the users behalf. If the user does not have to deal with the physical level, this is called the Physical Data Independence.

It is easy to use, queries are optimized automatically and it is isolated from changes to the physical schema.

Logical Data Independence

The set of tables personalized for a particular user is called the user's **external schema**. If users can be given their own external schema in a database system, it is told that this Database System supports Logical Data Independence.

It has three benefits:

- Each users gets a customized external schema, they see only the information they need.
- The user is isolated from changes to conceptual schema.
- It is safer.

1.2 Relational Databases

The relational model is a conceptual model since its schemas do not depend on the physical level.

1.2.1 Tables

The database is organized into **tables**, which contain zero or more **records** (ie: table rows), and at least one **fields** (ie: the columns of the table.) Each record has a value for each field, and all fields has a specific **type**. Often, when discussing tables, the type information ignored.

Null Values

A **null** value denotes a value that *does not exist* or is *unknown*. It occur if the data collection is incomplete or if data has not arrived yet.

1.2.2 Superkeys and Keys

In the relational model, the access to data is not handled by indices. Instead, a record must be referenced by specifying field values. Since not all values are guaranteed to be unique for all users, a unique identifier field is called a **superkey** to distinguish it. Adding a field to a superkey, will generate another superkey. A **key** is a superkey with the property that no subset of its fields is a super key.

Primary Keys

In the Schema at Figure 1.1's, **STUDENT** table **SI**d is a key. Whereas in **SECTION** there may be multiple keys if each professor teaches only one class. Therefore, since a table may have multiple keys, a key is chosen as a **Primary Key**, whose values *should never be null*, and who is used to refer to each record.

For instance, in Figure 1.1, **STUDENT** table, **SI**d can be the primary key. This is no coincidence, IDs are most times fit to be primary keys.

Foreign Keys

The information in a database is split among tables, these are not isolated from each other, a **foreign key** is a field (or fields) of one table which corresponds to the primary key of another table. For instance, in Schema at Figure 1.1, **CourseId** of the **SECTION** table is a foreign key.

Foreign Keys can be used to create logical connections between different types of records. In the Schema at Figure 1.1, **CourseId** of the **SECTION** table creates a logical connection between the **SECTION** table and **COURSE** table, since the objects these represent in real life, Sections and Courses are bound by a logical connection as well. (Each section is a section of a course).

Foreign Keys and Referential Integrity

The specification of a foreign key asserts **referential integrity**. Which requires each non-null foreign key value to be the key value of some record. Database system must ensure that if the primary keys of a table is modified in some ways, the foreign keys in other tables referring to primary keys must also be

updated accordingly, or set to **null** in worst case scenario.

1.2.3 Constraints

A **constraint** describes the allowable states that fields can have in a table. There are four important kinds of constraints. **Null Value Constraints** limit fields to not have null values. **Key constraints** specify that two records cannot have the same value. **Referential integrity constraints** specify referential integrity, finally **integrity constraints**.

Integrity constraints

These constraints encodes *business rules*. They can detect bad data entry and can enforce the *rules* of the organization. They may apply to tables, individual records or the entire database.

1.2.4 Table Specification in SQL

Listing 1.1: the SQL specification of the **STUDENT** table

```
create table STUDENT (
    SIId int not null,
    SName varchar(10) not null,
    MajorId int,
    GradYear int,

    primary key (SIId),
    foreign key (MajorId) references DEPT
        on update cascade
        on delete set null,
    check (SIId > 0),
    check (GradYear >= 1863)
)
```

In Listing 1.1 we can see constraints and fields. The action specified with the **on delete** and **on update** keywords can be one of the following:

Cascade causes the same query to apply to each foreign key record.

Set null causes the foreign key values to be set to null.

Set default causes the foreign key values to be set to their default value.

No action causes query to be rejected if there exists an affected value with the foreign key.