Class Notes - CS482 - Chapter 3

Questions/Issues

1. Why different levels of abstraction?
2. What is a data model?
3. What are the components of the relational data model?
4. Why constraints?
5. The Data Definition Language of SQL

Answering the above questions, you completed the Chapter 3 :-)

Question 1. Ease of development and maintenance of database applications.

The three levels of abstraction are physical (PL), conceptual (CL), and external level (EL). We can think of these levels as follows: PL – how data is stored, CL – how data is described; EL – how data is used. PL is the lowest level while EL is the highest one. These levels do not depend on each other:

CL does not depend on PL (*physical data independence*).

EL does not depend on CL (*conceptual data independence*).

How to access the stored data: let DBMS worry about it.

Conceptual schema is provided to application developers who use them to create programs accessing and presenting the data to users.

Question 2. A set of concepts, tools, and languages for describing

- CL and EL scheme (Data definition language — DDL)
- Constraints (DDL)
- Operations on relations (Data manipulation language — DML)

(Sometime, also a language for describing how the data should be stored — Storage definition language SDL)

Question 3. Relations and constraints.

Relation (or *Table*) – a set of tulples; consists of a relation schema and an instance

Instance – the concerete data that belongs to a relation at a particular time.

Relation schema: A relation schema is specified by a name, a set of attributes with their domain, and a set of constraints (more below), e.g., the student relation schema looks like:

```
Student(ID : integer, Name : String, Adress : String, Status : String)
```

with constraints such as ID must be integer; Status is either 'Freshman', 'Sophomore', 'Junior', 'Senior', and 'Graduate'; etc.

Constraints – statements that restrict the legal instances (called *static constraint*) or evolution of a relation database (called *dynamic constraint*);

Relational database: a finite set of relations and a set of constraints; *Database schema*: set of relation scheme; *Database instance*: set of corresponding relation instances.

Question 4. Constraints are necessary for the consistency of the data modeled by the database.

Constraints are enforced automatically by DBMS. (We ONLY need to specify them.) They can be used to maintain the consistency of the information (or database) or to enforce the rules of the enterprise.

Key. A subset $\bar{T}$ of attributes of a schema S is a key of S if

- for every instance I of S, no two rows in I have the same value with respect to $\bar{T}$;
- no proper subset of I has the above property.

Example: if every student has a different Id then $\{Id\}$ is a key of the relation schema STUDENT; on the other hand, $\{Id, Name\}$ is not a key since it has a proper subset $\{Id\}$ which satisfies the first property of the definition.

NOTE: Identifying a good set of keys (or at least one) is important in database design and analysis; every relation schema has at least one key; a relation schema might have more than one keys; key represents a *key constraint* on the relation; a key of a relation cannot be arbitrarily selected, it is the result of a careful consideration and analysis of the application.

Referential constraint. Requirement that *references tuples* must exist. Foreign key constraint is a special type of referential constraint in which the referenced attributes forms a key of the referenced relation.

the math's way: Given $S(\bar{F})$ and $T(\bar{K})$. It is *known* that each possible value of $\bar{T_1}$ in S (i.e., $\bar{T_1} \subseteq \bar{F}$ must exist as $\bar{T_2}$ in T (i.e., $\bar{T_2} \subseteq \bar{K}$. This indicates a referential constraint $S(\bar{T_1})$ references $T(\bar{T_2})$. If $\bar{T_2}$ is a key of T then we have a foreign key.

NOTE. Like keys, referential constraints (RC) need to be discovered during the analysis process; RC often involves more than one relations but there are situations where it involves only one relation.

Semantic constraints. Rules/conventions that restrict the evolution of the database.

DDL. The examples given below are Oracle's commands.

Create a table with one key:

```
CREATE TABLE STUDENT(
ID  NUMBER(10),
NAME  CHAR(10) NOT NULL,
     ADDRESS CHAR(50),
STATUS CHAR(10) DEFAULT freshman,
PRIMARY KEY(ID));
```

The last row can be replaced by `CONSTRAINT KEYCONSTRAINT PRIMARY KEY(ID));` which gives the *name* — KEYCONSTRAINT — for the constraint; the name can be used for the purpose of modifying or deleting the constraint.

Create a table with more than one key:

```
CREATE TABLE COURSE(
CrsCode CHAR(6),
DeptId CHAR(4),
CrsName CHAR(20),
Descr CHAR(100),
PRIMARY KEY (CrsCode),
UNIQUE (DeptId, CrsName));
```

The last row specifies that $\{DeptId, CrsName\}$ is another key of this relation.

Create a table with semantic constraints

```
CREATE TABLE Transcript (
     StudId  NUMBER(10),
     CrsCode  CHAR(6),
     Semester  CHAR(6),
     Grade  CHAR(1),
        FOREIGN KEY (STUDID) REFERENCES Student(ID),
     CHECK (Grade IN ('A', 'B', 'C', 'D', 'F', 'I')),
     CHECK (StudId > 0 AND StudId < 1000000000));
```

The last two rows specify two semantical constraints on this relation.

Adding a constraint: we can add a foreign key constraint to TRANSCRIPT as follows:

```
ALTER TABLE TRANSCRIPT
     ADD (CONSTRAINT TRANSCRIPT_01 FOREIGN KEY(CrsCode)
     REFERENCES Course(CrsCode));
```

Exersice 3.10 (Something simpler than what we developed in the class)

```
create table rentalitem (itemid char(10),
type char(2) not null,
descr char(100) not null,
primary key (itemid))

create table customer (customerid integer,
name char (20) not null,
address char(50) not null,
phoneno integer not null,
primary key (customerid),
check (phoneno > 999999 and phoneno < 10000000))

create table rentals(customerid integer,
  itemid char (10),
rentedfrom date,
renteduntil date,
datereturned date,
primary key (customerid, itemid, rentedfrom),
foreign key (customerid) references customer(customerid),
  foreign key (itemid) references item(itemid),
  check (rentedfrom <= renteduntil))
```

What if? What happens if

1. we delete a tuple from STUDENT?
2. we update the student id in a tuple of STUDENT?
3. we insert a tuple into TRANSCIPT and the student id does not exist in STUDENT?

Any of these activities can create a violation of the foreign key constraint `FOREIGN KEY(STUDID) REFERENCES STUDENT(ID)` in TRANSCRIPT. To enforce this constraints at all time, the DDL allows us to specify the actions that need to be taken when one of the above activity occurs. The DBMS will not allow us to execute the INSERTION but provides us with some options for other cases:

1. Cascading 'delete': if we delete a tuple from STUDENT then the referencing tuples will be deleted; This is specified by: `ON DELETE CASCADE`

   (this should followed the constraint `FOREIGN KEY STUDID) REFERENCES STUDENT(ID))`

2. Cascading 'update': if we update the student id in a tuple of STUDENT then the referencing tuples will be updated; This is specified by: `ON UPDATE CASCADE`

3. Set the values to be NULL or DEFAULT.

The following are also possible:

1. ON DELETE/UPDATE NO ACTION —- Cannot delete/update if referred to

2. ON DELETE/UPDATE SET NULL —- Set the value to NULL

3. ON DELETE/UPDATE SET DEFAULT —- Set the value of the attributes to default value