APAN PS5400: Managing Data

Week 4: Views, Functional Dependencies, Normal Forms

Lecturer: François Scharffe

COLUMBIA UNIVERSITY
School of Professional Studies

Recap of last week

- Creating relational databases using SQL
- Primary key of a table
- Foreign key constraints and referential integrity
- Other types of constraints
- Insertion of data into tables
- Modifying data
- Deleting tuples
- Using MySQL to create databases on your computer

This week

- Views
- Queries using views
- Functional dependencies
- BCNF
- Decomposition of tables to make database BCNF compliant (aka Normalization)

Views

- A view is a SQL query that is named and stored
    - Similar to a procedure in a programming language
- Similar to programing language procedures, views can be invoked by name and used to create other views and queries.
- Views are useful for exposing only some part of the data stored in some tables to users who should not be given access to all the data stored in those tables.
- For example, some employees may be permitted access to some part of the *instructor* table without access to the salary information stored in that table.

View Example

- General format:

  create view as view_name

  Any SQL query

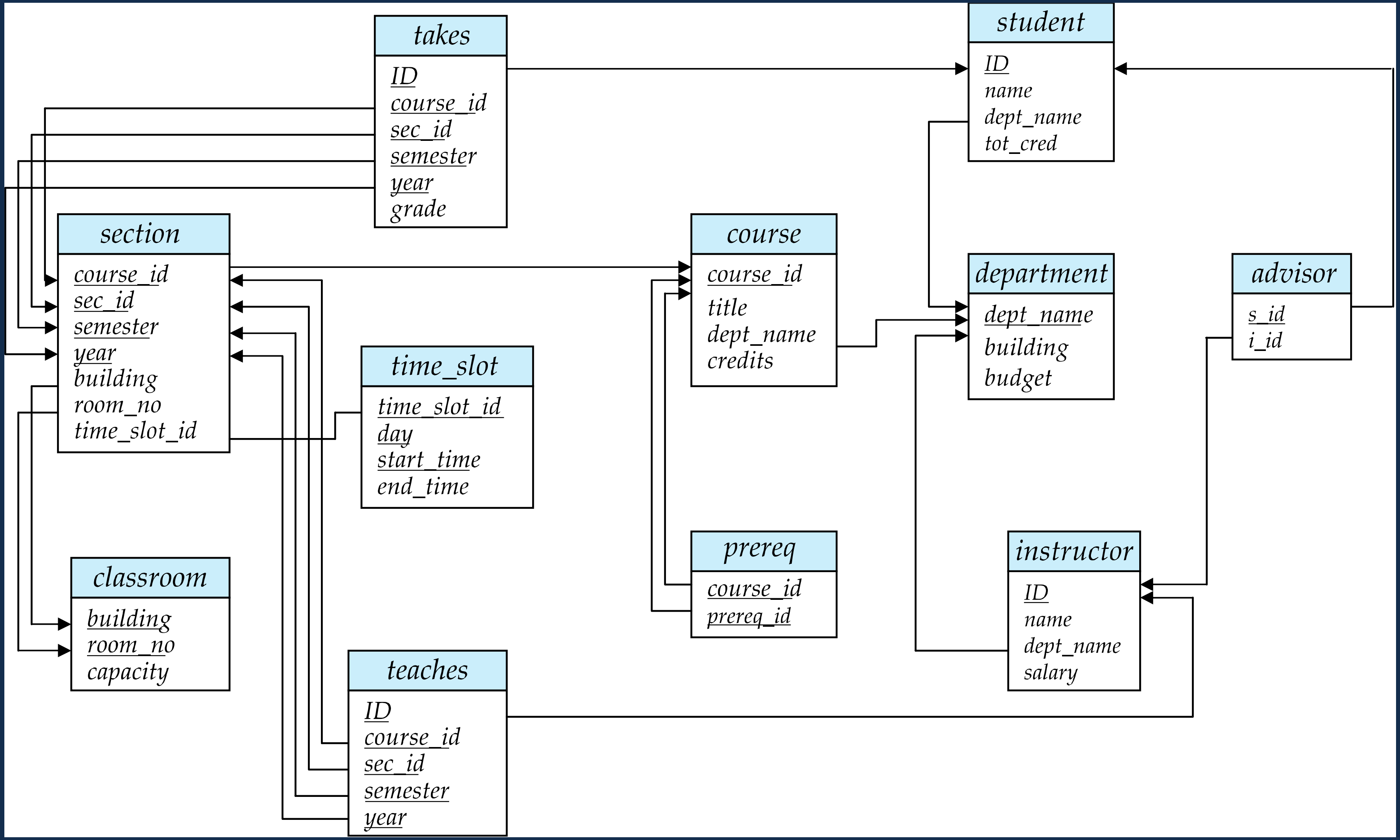
- To create a view on instructor table without *salary* information

  create view as no_salary_instructor

  select *id, name, dept_name*

  from instructor

Let us use this schema for the next few slides



From Database System Concepts, 6th Ed

COLUMBIA UNIVERSITY
School of Professional Studies

Let us create a view which provides information about which faculty advises which student without showing any confidential information such as ID and salary. We will use the previously created view *no_salary_information*

create view as *faculty_student_advising*
select *N.name AS InstructorName, N.dept_name AS InstructorDept,*
S.name AS StudentName, S.dept_name AS StudentDept
from student as S, advisor as A, no_salary_instructor as N
where S. ID = A.s_id AND N.id = A.i_id

view

Need to create aliases on ‘name’ and ‘dept-name’ so as to disambiguate between student info and instructor info.

Queries can also be created using views. Before execution of the query, it is expanded to include the definition of the view.

Find the building in which any instructor with the name ‘Kitty’ is housed

```
select  D.building
from department AS D, no_salary_instructor AS N
where D.dept_name = N.dept_name AND N.name LIKE “%Kitty%”
```

Functional Dependencies

- Functional dependencies can be used to encode the business rules of the organization for which the RDB has been created.
    - All employees of such and such rank are assigned to such and such parking lot.
    - Each department is assigned to a building and a budget
- They can also be used to provide additional semantics for attributes of tables.
    - Any student with junior (senior) standing must have completed 48 (72) number of credit hours
- Functional dependencies are also used to redesign RDBs

Functional dependencies continued

- The organization’s business rule “Every department must be assigned to a building and a budget” must be encoded in the DB for the organization and every valid instance of the DB must satisfy this business rule.
- This means no valid instance of the DB can contain two tuples with same department name and two different buildings (or budgets)
  - In this case we say that the functional dependency is *satisfied*
  - This is not to say that two different departments cannot be assigned to the same building.
- Functional dependencies are a kind of additional constraints on the database
- But an RDBMS does not directly enforce functional dependencies
  - Functional dependencies are enforced by a good design (or redesign) of a database
  - Typically, this means the attributes involved in a functional dependency are made to occur in a single table and constraints such as primary key or uniqueness constraints are used to enforce the functional dependency.

Formalisms

- Functional dependency $A \rightarrow X$, where A and X are subsets of attributes of a relation R.
  - The lhs (left hand side) of the dependency is A, the rhs is X
- This is a special case of a key K, which can be represented as $K \rightarrow R$ (all the attributes of R including K)
- Both of the above mean there can be no valid instance of the DB in which there are two tuples which have the same value for the lhs of the dependency but different values for the rhs of the dependency.
  - The main difference is that in the case of keys, the rhs consists of all the attributes of R.
  - Secondly, the DBMS has mechanisms for enforcing dependencies involving primary keys (i.e., primary key constraints)

FDs and Schema

Suppose that in our university schema instead of two separate tables, instructor and department, we had one table instructor-dept with the following schema:

instructor-dept (id, name, dept_name, salary, building, budget)

What is wrong with this schema?

id	name	dept_name	salary	building	budget
111	Jon Bergman	Biology	72,000	Watson	954,000
183	Ronald Sharma	Philosophy	68,000	Plaza	543,000
324	Seila Crick	Biology	73,000	Watson	954,000
379	Yuan Xi	Biology	64,000	Watson	954,000
421	Julie Fernandes	Philosophy	71,500	Plaza	543,000

Problems caused by FDs

id	name	dept_name	salary	building	budget
111	Jon Bergman	Biology	72,000	Watson	954,000
183	Ronald Sharma	Philosophy	68,000	Plaza	543,000
324	Seila Crick	Biology	73,000	Watson	954,000
379	Yuan Xi	Biology	64,000	Watson	954,000
421	Julie Fernandes	Philosophy	71,500	Plaza	543,000

- Redundancy anomaly
  - Building and budget info is needlessly repeated
  - Waste of DB space
- Insertion anomaly
  - If the philosophy budget is increased, it could be updated in one row and not the other rows.
- Deletion anomaly
  - If all the philosophy faculty are deleted from the DB, then there is no record of the location or budget of the philosophy department since there is no separate department table

Decomposition

- If we decompose instructor-dept into two schemas (as in our original DB design), instructor(id, name, dept_name) and department (dept_name, building, budget), all the redundancy problems go away.
    - The information about which department is in which building is now independent of the content of the instructor table
    - So this information is not lost even if instructor table has no row for any members of some department.
- The FD can be enforced by the DBMS by making dept_name the primary key in departments and making dept_name in instructor a foreign key reference to department

Problems with decomposition

- Decomposition needs to be done carefully otherwise it can generate its own problems
  - No information that is in the DB before decomposition should be lost as a result of decomposition
  - No information that is NOT in the DB before decomposition should be generated as a result of decomposition
    - When this condition is satisfied it is called a ‘lossless join decomposition’
    - We check whether the decomposition is a lossless join decomposition by doing a join of the decomposed schemas and checking if there are any new tuples that were not in the original table.
- Even if a decomposition is done carefully, there are problems
  - Joins are expensive and for many queries the system has to do a join on the decomposed tables
  - E.g., “Find the building in which Julie Fernandes has her office” would require a join of instructor and department tables.
  - Queries are more complex over a DB schemas with many decompositions.

FD: {dept, building} → P_lot

emp_id	dept	building	P_lot
11	Bio	Watson	A23
22	CS	Mudd1	B45
33	CS	Mudd2	B46

Decomposition into

A:

emp_id	dept
11	Bio
22	CS
33	CS

B:

dept	building	P_lot
Bio	Watson	A23
CS	Mudd1	B45
CS	Mudd2	B46

Join of A and B

emp_id	dept	building	P_lot
11	Bio	Watson	A23
22	CS	Mudd1	B45
22	CS	Mudd2	B46
33	CS	Mudd1	B45
33	CS	Mudd2	B46

Fake rows

Better Decomposition

emp-id	dept	building
11	Bio	Watson
22	CS	Mudd1
33	CS	Mudd2

dept	building	P_lot
Bio	Watson	A23
CS	Mudd1	B45
CS	Mudd2	B46

Closure of a set of FDs

- Given a set of FDs, ***F***, we can infer new FDs from ***F*** using Armstrong's Axioms
  - We will not be able to cover Armstrong's Axioms in this class
  - Thus, ***F*** can be expanded by adding these new FDs
- Possibly, more FDs can be added to ***F*** by inferring yet new FDs from the expanded set.
- The closure of ***F*** (***F^+***) is the set of FDs such that we can infer no new ones from the expanded set.
  - That is when ***F*** can be expanded no further through inference.
- A database is said to satisfy all the functional dependencies in ***F*** when it satisfies all the FDs in ***F^+*.**

Satisfaction of a set of FDs

- ***S*** satisfies ***F*** if every FD in ***F***$^{+}$ is satisfied by every relation in ***S.***
- Recall that a relation, R, in ***S*** satisfies a FD, F, if there are no two tuple in ***any*** instance of R such that they agree on the values of the lhs of F but differ on the values in the rhs of R.
- This implies that F *trivially* satisfies R if
  - R does not contain all the attributes on the lhs of F, or
  - R does not contain any of the attributes on the rhs of R, or
  - The attributes on the rhs of F are a subset of the lhs of F

How to tell whether FDs are satisfied

- Given a set of FDs, ***F***, and a DB schema, ***S***, how to tell whether ***S*** satisfies ***F***?
  - Recall that this is not the same as whether a given instance of ***S*** satisfies ***F.***
- In the university schema earlier with just one FD it was easy to tell that the FD {dept_name} → {building, budget}
  - was trivially satisfied by all the table except *department,* and
  - satisfied non-trivially in *department* because it contained all the attributes of the functional dependency and the lhs of the FD was a key of the table
- But when we have a large set of interacting FDs we need a more systematic approach
- Boyce-Codd Normal Form (BCNF) and Third Normal Form provide such an approach
  - We cover both BCNF and 3NF, with emphasis on 3NF.

Boyce-Codd Normal Form

- A database schema ***S*** is in BCNF with respect to a set of FDs ***F*** iff for every relation R in ***S*** and for every F in $\boldsymbol{F^+}$ it holds that

  1. rhs of F is a subset of lhs of F (so trivially satisfied), or
  2. lhs of F is a superkey of R (i.e., some key K of R is a subset of lhs of F)

- A database that is in BCNF with respect to ***F***
  - is guaranteed to satisfy all the FDs in ***F, and***
  - do it in such a way that there are no redundancy and update anomalies.

Third Normal Form (3NF)

- A database schema ***S*** is in 3NF with respect to a set of FDs ***F*** iff for every relation R in ***S*** and for every F in ***F***$^{+}$ it holds that

  1. rhs of F is a subset of lhs of F (so trivially satisfied), or
  2. lhs of F is a superkey of R (i.e., some key K of R is a subset of lhs of F), or
  3. Each attribute in rhs – lhs of R is contained in a candidate key of R (Note the attributes in rhs –lhs can be contained in different candidate keys of R.)

- Third Normal Form is a weakening of BCNF.

Normalization and 3NF

- A database schema that is not in 3NF can be put in 3NF (normalized wrt 3NF) through a process of decomposing some of the schema of the database.
- Only those schema in the database schema need to be normalized which are not in 3NF
- But not all ways of decomposing a schema will lead to the database being in 3NF.
- There are algorithms for decomposing schema which guarantee that the final decomposed schema is in 3NF
    - They also guarantee that the decomposition is a lossless join decomposition (the join of any decomposed relation will result in the original relation)
    - And it is dependency preserving (dependencies can be checked again the decompositions without having to re-create the original schema).
    - We will not be covering these algorithms

Example of Decomposition

- Suppose we have a relation with attributes {S,N,L,R,W,H} with S as its only key
- We have two FDs: S→ {N,L} and R → W
- S → {N,L} is satisfied because its lhs contains the key
- R → W is not satisfied. So we need to decompose the relation.
- The relation can be decomposed into two relations:
  - {S, N, L, R, H} and {R, W}
  - These two relations satisfy both FDs
  - Why?

In the time remaining we will discuss project 1.2.

Recap: What we covered this week

- Views
- Functional dependencies
- Normal Forms (BCNF and 3NF)
- Normalization
- Project 1.2