Databases

Lecture 5

Functional Dependencies. Normal Forms (II)

Sabina S. CS

- *simple attribute*
- *composite attribute* - a set of attributes in a relation (with at least two attributes)

- in some applications, attributes (simple or composite) can take on multiple values for a record in the relation - *repeating* attributes
- when defining a relation (in the relational model), attribute values must be scalar, atomic (they cannot be further decomposed)

Example 2. Consider the relation:

STUDENT [NAME, BIRTHYEAR, GROUP, COURSE, GRADE]

- key: NAME
- *composite repeating attribute:* the pair {COURSE, GRADE}
- this relation could store the following values:

NAME	BIRTHYEAR	GROUP	COURSE	GRADE
Ioana	2000	921	Databases Distributed Operating Systems Probabilities and Statistics	10 9.99 10
Vasile	2000	925	Databases Distributed Operating Systems Probabilities and Statistics Logical and Functional Programming	10 9.98 10 10

Example 3. Consider the relation:

BOOK [BId, AuthorsNames, Title, Publisher, PublishingYear, Keywords]

- key: BId
- *simple repeating attributes*: AuthorsNames, Keywords
- the BId attribute
  - could have an actual meaning (an id could be associated with each book)
  - could be introduced as a key (with distinct values that are automatically generated)

- repeating attributes cannot be used in the relational model, hence they are to be avoided, without losing data
- let $R[A]$ be a relation; A - the set of attributes
- let α be a repeating attribute in R (simple or composite)
- R can be decomposed into 2 relations, such that α is not a repeating attribute anymore
- if K is a key in R, the two relations into which R is decomposed are:

$$R'[K \cup \alpha] = \Pi_{K \cup \alpha}(R)$$
$$R''[A - \alpha] = \Pi_{A-\alpha}(R)$$

Example 4. The STUDENT relation from example 2 is decomposed into the following 2 relations:

GENERAL_DATA [NAME, BIRTHYEAR, GROUP],

RESULTS [NAME, COURSE, GRADE].

- the values in these relations are:

NAME	BIRTHYEAR	GROUP
Ioana	2000	921
Vasile	2000	925

NAME	COURSE	GRADE
Ioana	Databases	10
Ioana	Distributed Operating Systems	9.99
Ioana	Probabilities and Statistics	10
Vasile	Databases	10
Vasile	Distributed Operating Systems	9.98
Vasile	Probabilities and Statistics	10
Vasile	Logical and Functional Programming	10

Example 5. The BOOK relation in example 3 is decomposed into the following 3 relations (there are two repeating attributes in the BOOK relation):

BOOKS [BId, Title, Publisher, PublishingYear],

AUTHORS [BId, AuthorName],

KEYWORDS [BId, Keyword].

- is a book is not associated with any authors / keywords, it must have one corresponding tuple in AUTHORS / KEYWORDS, with the second attribute set to *null*; in the absence of such tuples, the BOOK relation can't be obtained from the three relations using just the natural join (outer join operators are required)

Definition. A relation is in the first normal form (1NF) if and only if it doesn't have any repeating attributes.

- obs. some DBMSs can manage non-1NF relations (e.g., Oracle)

- the following 3 relational normal forms use a very important notion: the *functional dependency* among subsets of attributes
- the database administrator is responsible with determining the functional dependencies; these depend on the nature, the semantics of the data stored in the relation

->

Definition. Let $R[A_1, A_2, \ldots, A_n]$ be a relation and α, β two subsets of attributes of R. The (simple or composite) attribute α functionally determines attribute β (simple or composite), notation:

$$\alpha \rightarrow \beta$$

if and only if every value of α in R is associated with a precise, unique value for β (this association holds throughout the entire existence of relation R); if an α value appears in multiple rows, each of these rows will contain the same value for β:

$$\Pi_\alpha(r) = \Pi_\alpha(r') \ \ implies \ \ \Pi_\beta(r) = \Pi_\beta(r')$$

- in the dependency $\alpha \rightarrow \beta$, α is the *determinant*, and β is the *dependent*
- the functional dependency can be regarded as a property (restriction) that must be satisfied by the database throughout its existence: values can be added / changed in the relation only if the functional dependency is satisfied

- if a relation contains a functional dependency, associations among values can be stored multiple times (data redundancy)
- to describe some of the problems that can arise in this context, we'll consider the EXAM relation, storing students' final exam results

Example 6. EXAM [StudentName, Course, Grade, FacultyMember]

- key: {StudentName, Course}
- a course is associated with one faculty member, a faculty member can be associated with several courses; the following restriction (functional dependency) must be satisfied: *{Course}* → *{FacultyMember}*

EXAM	StudentName	Course	Grade	FacultyMember
1	Pop Ioana	Computer Networks	10	Matei Ana
2	Vlad Ana	Operating Systems	10	Simion Bogdan
3	Vlad Ana	Computer Networks	9.98	Matei Ana
4	Dan Andrei	Computer Networks	10	Matei Ana
5	Popescu Alex	Operating Systems	9.99	Simion Bogdan

- if the relation contains such a functional dependency, the following problems can arise (some of them need to be solved through additional programming efforts - executing a SQL command is not enough):
  - wasting space: the same associations are stored multiple times, e.g., the one among *Computer Networks* and *Matei Ana* is stored 3 times;
  - update anomalies: if the *FacultyMember* is changed for course *c*, the change must be carried out in all associations involving course *c* (without knowing how many such associations exist), otherwise the database will contain errors (it will be inconsistent); if the *FacultyMember* value is changed in the 2nd record, but not in the 5th record, the operation will introduce an error in the relation (the data will be incorrect);
  - insertion anomalies: one cannot specify the *FacultyMember* for a course *c*, unless there's at least one student with a grade in course *c*;

- deletion anomalies: when some records are deleted, data that is not intended to be removed can be deleted as well; e.g., if records 1, 3 and 4 are deleted, the association among *Course* and *FacultyMember* is also removed from the database
- the functional dependency among sets of attributes is what causes the previous anomalies; to eliminate these problems, the associations (dependencies) among values should be kept once in a separate relation, therefore, the initial relation should be decomposed (via a good decomposition - data should not be lost or added); such a decomposition is performed in the database design phase, when functional dependencies can be identified

Obs. The following simple properties for functional dependencies can be easily demonstrated:

1. If *K* is a key of $R[A_1, A_2, \ldots, A_n]$, then $K \rightarrow \beta$, $\forall \beta$ a subset of $\{A_1, A_2, \ldots, A_n\}$.

- such a dependency is always true, hence it will not be eliminated through decompositions

2. If $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ - *trivial functional dependency* (*reflexivity*).

$$\Pi_\alpha(r_1) = \Pi_\alpha(r_2) \Rightarrow \Pi_\beta(r_1) = \Pi_\beta(r_2) \Rightarrow \alpha \rightarrow \beta$$
$$\beta \subseteq \alpha$$

3. If $\alpha \rightarrow \beta$, then $\gamma \rightarrow \beta$, $\forall \gamma$ with $\alpha \subset \gamma$.

$$\Pi_\gamma(r_1) = \Pi_\gamma(r_2) \underset{\alpha \subset \gamma, prop.2}{\Rightarrow} \Pi_\alpha(r_1) = \Pi_\alpha(r_2) \underset{\alpha \rightarrow \beta}{\Rightarrow} \Pi_\beta(r_1) = \Pi_\beta(r_2) \Rightarrow \gamma \rightarrow \beta$$

Obs. The following simple properties for functional dependencies can be easily demonstrated:

4. If $\alpha \to \beta$ and $\beta \to \gamma$, then $\alpha \to \gamma$ - *transitivity*.

$$\Pi_{\alpha}(r_1) = \Pi_{\alpha}(r_2) \underset{\alpha \to \beta}{\Rightarrow} \Pi_{\beta}(r_1) = \Pi_{\beta}(r_2) \underset{\beta \to \gamma}{\Rightarrow} \Pi_{\gamma}(r_1) = \Pi_{\gamma}(r_2) \Rightarrow \alpha \to \gamma$$

5. If $\alpha \to \beta$ and γ a subset of $\{A_1, \dots, A_n\}$, then $\alpha\gamma \to \beta\gamma$, where $\alpha\gamma = \alpha \cup \gamma$.

$$\Pi_{\alpha\gamma}(r_1) = \Pi_{\alpha\gamma}(r_2) \Rightarrow \left| \begin{matrix} \Pi_{\alpha}(r_1) = \Pi_{\alpha}(r_2) \Rightarrow \Pi_{\beta}(r_1) = \Pi_{\beta}(r_2) \\ \Pi_{\gamma}(r_1) = \Pi_{\gamma}(r_2) \end{matrix} \right| \Rightarrow \Pi_{\beta\gamma}(r_1) = \Pi_{\beta\gamma}(r_2)$$

Definition. An attribute A (simple or composite) is said to be *prime* if there is a key K and A ⊆ K (K can be a composite key; A can itself be a key). If an attribute isn't included in any key, it is said to be *non-prime*.

Definition. Let $R[A_1, A_2, \ldots, A_n]$ be a relation, and let α, β be two subsets of attributes of R. Attribute β is *fully functionally dependent on* α if:

- β is functionally dependent on α (i.e., $\alpha \rightarrow \beta$) and
- β is not functionally dependent on any proper subset of α, i.e., $\forall\, \gamma \subset \alpha$, $\gamma \rightarrow \beta$ does not hold.

Definition. A relation is in the second normal form (2NF) if and only if:

1. it is in the first normal form and

2. every (simple or composite) non-prime attribute is fully functionally dependent on every key of the relation.

- obs. Let R be a 1NF relation that is not 2NF. Then R has a composite key (and a functional dependency $\alpha \to \beta$, where α (simple or composite) is a proper subset of a key and β is a non-prime attribute).
- decomposition
  - relation $R[A]$ (A - the set of attributes), K - a key
  - β non-prime, β functionally dependent on α, $\alpha \subset K$ (β is functionally dependent on a proper subset of attributes from a key)
  - the $\alpha \to \beta$ dependency can be eliminated if R is decomposed into the following 2 relations:

$$R'[\alpha \cup \beta] = \Pi_{\alpha \cup \beta}(R)$$
$$R''[A - \beta] = \Pi_{A-\beta}(R)$$

- we'll analyze the relation from Example 6

EXAM[StudentName, Course, Grade, FacultyMember]

- key: {StudentName, Course}
- the functional dependency *{Course}* → *{FacultyMember}* holds => attribute *FacultyMember* is not fully functionally dependent on a key, hence the EXAM relation is not in 2NF
- this dependency can be eliminated if EXAM is decomposed into the following 2 relations:

RESULTS[StudentName, Course, Grade]

COURSES[Course, FacultyMember]

Example 7. Consider the following relation, storing students' learning contracts:

CONTRACTS[LastName, FirstName, CNP, CourseId, CourseName]

- key: {CNP, CourseId}
- functional dependencies: *{CNP}* → *{LastName, FirstName}*, *{CourseId}* → *{CourseName}*
- to eliminate these dependencies, the relation is decomposed into the following three relations:

STUDENTS[CNP, LastName, FirstName]

COURSES[CourseId, CourseName]

LEARNING_CONTRACTS[CNP, CourseId]

- the notion of *transitive dependency* is required for the third normal form

Definition. An attribute *Z* is transitively dependent on an attribute *X* if $\exists Y$ such that $X \to Y$, $Y \to Z$, $Y \to X$ does not hold (and *Z* is not in X or Y).

Definition. A relation is in the third normal form (3NF) if and only if it is in the second normal form and no non-prime attribute is transitively dependent on any key in the relation.

Another definition: A relation R is in the third normal form (3NF) iff, for every non-trivial functional dependency $X \to A$ that holds over R:

- X is a superkey, or
- A is a prime attribute.

Example 8. The BSc examination results are stored in the relation:

BSC_EXAM [StudentName, Grade, Supervisor, Department]

- the relation stores the supervisor and the department in which she works
- since the relation contains data about students (i.e., one row per student), *StudentName* can be chosen as the key
- the following functional dependency holds: $\{Supervisor\} \rightarrow \{Department\} \Rightarrow$ the relation is not in 3NF
- to eliminate this dependency, the relation is decomposed into the following 2 relations:

RESULTS [StudentName, Grade, Supervisor]

SUPERVISORS [Supervisor, Department]

Example 9. The following relation stores addresses for a group of people:

ADDRESSES [CNP, LastName, FirstName, ZipCode, City, Street, No]

- key: {CNP}
- identified dependency: *{ZipCode}* → *{City}* (can you identify another dependency in this relation?)
- since this dependency holds, relation ADDRESSES is not in 3NF, therefore it must be decomposed:

ADDRESSES'[CNP, LastName, FirstName, ZipCode, Street, No]

ZIPCODES[ZipCode, City]

Example 10. The following relation stores the exam session schedule:

EX_SCHEDULE[Date, Hour, Faculty_member, Room, Group]

- the following restrictions are expressed via key definitions and functional dependencies:

    1. a group of students has at most one exam per day

    => *{Date, Group}* is a key

    2. on a certain date and time, a faculty member has at most one exam

    => *{Faculty_member, Date, Hour}* is a key

    3. on a certain date and time, there is at most one exam in a room

    => *{Room, Date, Hour}* is a key

    4. a faculty member doesn't change the room in a day

    => the following dependency holds: $\{Faculty\_member, Date\} \rightarrow \{Room\}$

- all attributes appear in at least one key, i.e., there are no non-prime attributes
- given the normal forms' definitions specified thus far, the relation is in 3NF
- objective: eliminate the *{Faculty_member, Date}* → *{Room}* functional dependency

Definition. A relation is in the Boyce-Codd (BCNF) normal form if and only if every determinant (for a functional dependency) is a key (informal definition - simplifying assumption: determinants are not too big; only non-trivial functional dependencies are considered).

- to eliminate the functional dependency, the original relation must be decomposed into:

EX_SCHEDULE'[Date, Hour, Faculty_member, Group],

ROOM_ALLOCATION[Faculty_member, Date, Room]

- these relations don't contain other functional dependencies, i.e., they are in BCNF
- however, the key associated with the 3rd constraint, *{Room, Date, Hour}*, does not exist anymore
- if this constraint is to be kept, it needs to be checked in a different manner (e.g., through the program)