GALGOTIAS UNIVERSITY

School of Computing Science and Engineering

Program: BCA

Course Code:BCAC2102

Course Name: Database Management System

Lecture-16

Topic- Functional Dependencies

Faculty:-Dr. Satyajee Srivastava

Lecture-15(RECAP)

Topic- Relational database design: pitfalls

Objective :

Discuss Relational database design: pitfalls.

Lecture-15

RELATIONAL DATABASE DESIGN

Basic Concepts

- a **database** is an collection of logically related *records*
- a **relational database** stores its data in 2-dimensional *tables*
- a **table** is a two-dimensional structure made up of *rows* (*tuples*, *records*) and *columns* (*attributes*, *fields*)
- example: a table of students engaged in sports activities, where a student is allowed to participate in at most one activity

StudentID	Activity	Fee
100	Skiing	200
150	Swimming	50
175	Squash	50
200	Swimming	50

Lecture-15

What is Relational Algebra?

Relational algebra is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operation to perform this action.

Relational algebra operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.

Lecture-15

Basic Relational Algebra Operations:

Relational Algebra devided in various groups

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol:)

Lecture-15

Relational Algebra Operations From Set Theory

- UNION (u)
- INTERSECTION ( ),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Binary Relational Operations

- JOIN
- DIVISION

Lecture-15 SELECT (σ)

- $\sigma_p(r)$
- σ is the predicate
- r stands for relation which is the name of the table
- p is prepositional logic
- **Example 1**
- $\sigma_{topic = "Database"}(Subject)$

Output - Selects tuples from Subject where topic = 'Dat abase'.

Lecture-15

Example 2

- $\sigma_{\text{topic = "Database" and author = “korth"}}$(Subject)
- **Output** - Selects tuples from Tutorials where the topic is 'Database' and 'author' is korth.
- **Example 3**
- $\sigma_{\text{sales > 50000}}$ (Customers)
- **Output** - Selects tuples from Customers where sales is greater than 50000

Lecture-15

Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminates duplicate values. (pi) The symbol used to choose attributes from a relation. This operation helps you to keep specific columns from a relation and discards the other columns.

Lecture-15

Example of Projection:

Consider the following table

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Here, the projection of CustomerName and status will give

$$\Pi_{\text{CustomerName, Status}}(\text{Customers})$$

Lecture-15

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Lecture-15

Operation	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference(-)	- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.
Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.

Lecture-15

Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.

Lecture-15

EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(⟖)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(⟗)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition

Relational database design: pitfalls

Redundant storage of data:

Office Phone & HOD info - stored redundantly

- once with each student that belongs to the department
- wastage of disk space

A program that updates Office Phone of a department

- must change it at several places
  - more running time
  - error - prone

Transactions running on a database

- must take as short time as possible to increase transaction throughput

Relational database design: pitfalls

Update Anomalies

Another kind of problem with bad schema

Insertion anomaly:

No way of inserting info about a new department unless we also enter details of a (dummy) student in the department

Deletion anomaly:

If all students of a certain department leave and we delete their tuples, information about the department itself is lost

Relational database design: pitfalls

Update Anomaly:

Updating officePhone of a department

- value in several tuples needs to be changed
- if a tuple is missed - inconsistency in data

Lecture-15

(Assignment)

Discuss Relational database design: pitfalls.

Lecture-16

Topic- Functional Dependencies

Objective :

Functional Dependencies and their properties

Lecture-16

Consider Two table:- 1.

STATE table:

State Abbrev	StateName	Union Order	StateBird	State Population
CT	Connecticut	5	American robin	3,287,116
MI	Michigan	26	robin	9,295,297
SD	South Dakota	40	pheasant	696,004
TN	Tennessee	16	mocking bird	4,877,185
TX	Texas	28	mocking bird	16,986,510

Lecture-16

Consider Two table:- 2.

CITY table:

State Abbrev	CityName	City Population
CT	Hartford	139,739
CT	Madison	14,031
CT	Portland	8,418
MI	Lansing	127,321
SD	Madison	6,257
SD	Pierre	12,906
TN	Nashville	488,374
TX	Austin	465,622
TX	Portland	12,224

Lecture-16

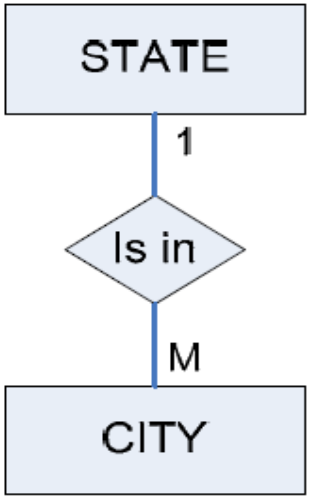
Outline Notation

STATE(StateAbbrev, StateName, UnionOrder,
StateBird, StatePopulation)

CITY(StateAbbrev, CityName, CityPopulation)
StateAbbrev foreign key to STATE

- Underline all parts of each primary key
- Note foreign keys with "*attribute* foreign key to *TABLE*"

Lecture-16



- one-to-many relationships: to determine the direction, always start with "one"
  - "*one* city is in *one* state"
  - "*one* state contains *many* cities"
- the foreign key is always in "the many" – otherwise it could not be atomic (it would have to be a list)

Lecture-16

Functional Dependency

- attribute B is functionally dependent on attribute A if given a value of attribute A, there is only one possible corresponding value of attribute B
  - that is, any two rows with the same value of A must have the same value for B
- attribute A is the determinant of attribute B if attribute B is functionally dependent on attribute A
  - in the STATE relation above, StateAbbrev is a determinant of all other attributes
  - in the STATE relation, the attribute StateName is also a determinant of all other attributes
  - so, StateAbbrev and StateName are both candidate keys for STATE

Lecture-16

- in the CITY relation above, the attributes (StateAbbrev, CityName) together are a determinant of the attribute CityPopulation
- in the CITY relation, the attribute CityName is not a determinant of the attribute CityPopulation because multiple cities in the table may have the same name

Lecture-16

- in the CITY relation above, the attributes (StateAbbrev, CityName) together are a determinant of the attribute CityPopulation
- in the CITY relation, the attribute CityName is not a determinant of the attribute CityPopulation because multiple cities in the table may have the same name

Lecture-16

- Functional dependencies

Functional dependencies are a constraint on the set of legal relations.

Let, $\alpha \subseteq R$ and $\beta \supseteq R$. The functional dependency $\alpha \rightarrow \beta$ holds on R if in any legal relation r(R). For all pairs of tuples t_1 and t_2 in r such that $t_1(\alpha) = t_2(\alpha)$

Functional dependencies provide a means for defining additional constraints on a relational schema. In simple words, a tuple value in one attribute uniquely determines the tuple's value in another attribute.

Example:

WORKER–ID uniquely determines NAME and WORKER–ID uniquely determines SKILL–TYPE, therefore functional dependencies as

FD : WORKER–ID → NAME

FD : WORKER–ID → SKILL–TYPE

The notation "→" is read "functionally determines".

Lecture-16

- Functional dependencies

Thus, in these examples, WORKER-ID functionally determines NAME, WORKER-ID functionally determines SKILL-TYPE.

The attribute on the left hand side of an FD is called a determinant because its value determines the value of the attribute on right-hand side. A relation's key is a determinant, since its value uniquely determines the value of every attribute in a tuple.

(1) Functional dependency of two attributes :

Consider Branch relation

Branch (Branch-name, branch-city, assets) on Branch-schema.

Branch-name → branch-city

Branch-name → assets

Lecture-16

- Functional dependencies

(2) Example for No functional dependencies :

Consider Depositor–schema relation

Depositor (customer–name, Account–number)

Here, customer–name and Account–number together form primary key.

Customer–name and Account–number are foreign keys.

Therefore no functional dependencies.

Lecture-16

- Functional dependencies

(3) Example of a relation that has a functional dependency in which the determinant has two or more attributes.
Consider STUDENT_COURSE_INFO relation
xSTUDENT_COURSE_INFO (Name, Course, Grade, Phone–no., Major, Course–Dept)

Name → Phone
Course →Course–Dept
Name course → Grade

Name course is a candidate key
Name & Course are prime attributes.
Grade is fully functionally dependent on the candidate key.
Phone–no, Course–Dept and major are partially dependent on the candidate key.

Lecture-16

- Functional dependencies

(4) Transitive and Trivial Functional dependencies :

Transitive Functional Dependencies :

It occurs when a non key attribute is functionally dependent on one or more other non key attributes.

A functional dependency $X \rightarrow Y$ in a relation scheme & is a transitive dependency if there is a set of attributes Z that is neither a candidate key nor a subset if any key of R and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold.

Trivial Functional dependencies :

Let R be a relation on the relation schema R, then R satisfies the functional dependency $X \rightarrow Y$ if a given set of values for each of the values of the attribute in X uniquely determines each of the values of the attributes in Y. Y is said to be functionally dependent on X. The functional dependency is denoted as $X \rightarrow Y$, where X is the left hand side or the determinant of the R and Y is the right hand side of the FD.

A functional dependency $X \rightarrow Y$ is said to be trivial if $Y \subseteq X$ or $Y \subseteq X$, $X \rightarrow Y$.

A functional dependency $X \rightarrow Y$ is said to be trivial functional dependency if $Y \subseteq X$.

Lecture-16

- Functional dependencies

  - A functional dependency is **trivial** if it is satisfied by all instances of a relation
    - Example:
      - *ID, name* $\rightarrow$ *ID*
      - *name* $\rightarrow$ *name*
    - In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Lecture-16

Closure of a Set of Functional Dependencies

- Given a set F of functional dependencies, there are certain other functional dependencies that are logically implied by F.
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by F is the **closure** of F.
- We denote the *closure* of F by $\mathbf{F^+}$.
- F^+ is a superset of F.

Lecture-16

The closure of F, denoted by F^+, is the set of all functional dependencies logically implied by F.

The closure of F can be found by using a collection of rules called **Armstrong axioms.**

Reflexivity rule: If A *is a set of attributes and B is subset or equal to A, then* A→B holds.

Augmentation rule: If A→B holds and C is a set of attributes, then CA→CB holds

Transitivity rule: If A→B holds and B→C holds, then A→C holds.

Union rule: If A→B holds and A→C then A→BC holds

Decomposition rule: If A→BC holds, then A→B holds and A→C holds.

Pseudo transitivity rule: If A→B holds and BC→D holds, then AC→D holds.

Lecture-16

(Assignment)

Explain the classification of functional dependency.

Thank You