functional Dependency

Tuesday, 18 June 2024 2:35 AM

Database design →

① How do we design a 'good database' schema?

↳ Here we are talking about high level logical blueprint, not about what DS to use, that's more of physical design.

So you the application designer need to design the db on the logical level.

To ensure goodness→

① We want to ensure we don't lose any data (db integrity)

② Good performance.

Let's take an example db→

student(sid,cid,room,grade,name,address)					
sid	cid	room	grade	name	address
123	15-445	GHC 6115	A	Andy	Pittsburgh
456	15-721	GHC 8102	B	Tupac	Los Angeles
789	15-445	GHC 6115	A	Obama	Chicago
012	15-445	GHC 6115	C	Waka Flocka	Atlanta
789	15-721	GHC 8102	A	Obama	Chicago

Any Problems?

① Data Duplication → on address, room and name

due to this redundancy some anomalies can occur.

a) Update Anomalies→ if room no changes, we need to change all student record with that room no.

b) Insert Anomalies → May not be possible to add a student unless they're enrolled in a course.

c) delete anomalies → If all students enrolled in a course are deleted, we lose the room no.

Finally we decompose that giant single relation in sub relations & link them.

student(sid,name,address)		
sid	name	address
123	Andy	Pittsburgh
456	Tupac	Los Angeles
789	Obama	Chicago
012	Waka Flocka	Atlanta

rooms(cid,room)	
cid	room
15-415	GHC 6115
15-721	GHC 8102

courses(sid,cid,grade)		
sid	cid	grade
123	15-415	A
456	15-721	B
789	15-415	A
012	15-415	C
789	15-721	A

decomposition

We need to dig out exactly why this is better??

★ Functional dependency

It is a form of constraint, that defines that the value of one attribute is implied by the other.

Definition → $X \to Y$ → Indicates X tend to be primary key

↳ The value of X functionally defines value of Y

Formal definition → for 2 tuples

$X \to Y \implies (t_1[x] = t_2[x] \implies t_1[y] = t_2[y])$

if 2 tuples (t_1, t_2) agree on X attribute then they must agree on Y too.

R1(sid,name,address)		
sid	name	address
123	Andy	Pittsburgh
456	Tupac	Los Angeles
789	Obama	Chicago
012	Waka Flocka	Atlanta
555	Andy	Providence

Note→ Two FDs $X \to Y$ and $X \to Z$ can be written as $X \to YZ$

But $XY \to Z$ is not same as $X \to Z$ and $Y \to Z$

Defining FDs in SQL

If we define FD as → FD: sid → name

```
CREATE ASSERTION student-name
  CHECK (NOT EXISTS
    (SELECT * FROM students AS s1,
                   students AS s2
     WHERE s1.sid = s2.sid
       AND s1.name <> s2.name))
```

→ for any operation on student table we would like to run this

Note→ It's in the SQL standard - 1992, but as of now (2024) no Database supports SQL-92 assertions.

Oracle considered to build it, but no update trigger. Although DB2 has some thing close using constraints.

```
CREATE TABLE students (
  sid INT PRIMARY KEY,
  name VARCHAR(32),
  ⋮
  CONSTRAINT student_name CHECK (name)
    DETERMINED BY (sid)
);
```

IBM

→ limited to single attributes

Q: why if SQL language doesn't exist, why we should care?

We need them to check if our design is good in terms of integrity or not.

★ Implied dependencies

Provided FDs → sid → name, address

sid, cid → grade

Implied FDs → sid, cid → grade

sid, cid → sid

Closure → (F^+)

Explosion of all possible implied FDs given our FDs

We use armstrong's axioms to use this

Reflexivity → $X \supseteq Y \implies X \to Y$

Augmentation → $X \to Y \implies XZ \to YZ$

Transitivity → $(X \to Y) \wedge (Y \to Z) \implies X \to Z$

Union → $(X \to Y) \wedge (X \to Z) \implies X \to YZ$

Decomposition → $X \to YZ \implies (X \to Y) \wedge (X \to Z)$

Pseudo transitivity → $(X \to Y) \wedge (YW \to Z) \implies XW \to Z$

Q: why do we need closure

With closure we can compute the attribute closure:

for a given attribute X, the closure X^+ is set of all attributes such that $X \to A$ can be inferred using the axioms.

To check $X \to A$

find X^+

Check → $A \in X^+$

BUT AGAIN, WHY SHOULD I CARE?

Maintaining the closure at runtime is expensive:

→ The DBMS has to check all the constraints for every INSERT, UPDATE, and DELETE operation.

We want a minimal set of FDs that was enough to ensure correctness.

We need a min set of FDs that's enough to ensure correctness. This min set is called Canonical cover.

Properties of Canonical Cover (F_c)

① RHS of every FD is a single attribute

② Closure of F_c is identical to closure of F.

③ F_c is minimal

Example

$AB \to C$
$A \to BC$
$B \to C$
$A \to B$

→

$AB \to C$
$A \to B$
$A \to C$
$B \to C$
$A \to B$

→ (simplify as per ①)

$AB \to C$
$A \to B$
$B \to C$

→

$A \to B$
$B \to C$

Canonical cover allows us to find Super key -

Super key

Any set of attributes in a relation that functionally determines all attributes in a relation

Candidate key

Any super key that the removal of any attribute leaves a set that does not functionally determine all attributes

Super key are a set of attributes that uniquely identifies a tuple

minimal set of super key is candidate key.

Why care about Super keys -

Helps us to determine whether it is okay to decompose a table into multiple tables.

Ensures that we are able to reconstruct original relation through joins.

→ Lossless decomposition

→ Dependency preservation → if it's original FDs don't span multiple tables.

Eg $R_1(A,B,C)$ $R_2(C,D)$

$F = \{A \to B, AB \to C, C \to D\}$ NO

Eg $R_1(A,B,D)$ $R_2(C,D)$

$F = \{A \to B, AB \to D, C \to D\}$ Yes

→ Redundancy avoidance.