Relational Database Design

Functional Dependencies

October 2023

Informal Design Guidelines for Relational Databases

- What is relational database design?
  - The grouping of attributes to form "good" relation schemas
- Design is concerned mainly with base relations
- What are the criteria for "good" base relations?
- Let’s first discuss informal guidelines for good relational design.

Semantics of the Relation Attributes

- GUIDELINE 1: Informally, each tuple in a relation should represent one entity or relationship instance.
  - Attributes of different entities (EMPLOYEEs, PROJECTs etc.) should not be mixed in the same relation
  - Foreign keys should be used to refer to other entities
  - Entity and relationship attributes should be kept apart as much as possible.
- Key point: Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.

Redundant Information in Tuples

- The Evils of Redundancy
  - Wastes storage
  - Causes problems with update anomalies
    - Insertion anomalies
    - Deletion anomalies
    - Modification anomalies

EXAMPLE: AN UPDATE ANOMALY

- Consider a table: student_activity

SID	*Activity*	Place	Fee
100	Skiing	Mt. Dora	200.00
100	Golf	Orlando Links	50.00
150	Swimming	Lake Beatrice	100.00
175	Squash	UCF Courts	100.00
175	Swimming	Lake Beatrice	100.00

Note that:
Activity ---> Place
and
Activity ---> Fee

- If SID 100 drops golf, we lose the cost of golf. (Delete anomaly)
- If the fee for swimming changes, it must be changed in several places. (Update anomaly)
- The cost of an activity may not be added until a student participates in it. (Insertion anomaly)

Solution of UPDATE ANOMALY

- Solution ??

- Main refinement technique:
  - *decomposition*
  - Example:
    - Replacing ABCD with, say, AB and BCD

EXAMPLE: AN UPDATE ANOMALY

SID	***Activity***	Place	Fee
100	Skiing	Mt. Dora	200.00
100	Golf	Orlando Links	50.00
150	Swimming	Lake Beatrice	100.00
175	Squash	UCF Courts	100.00
175	Swimming	Lake Beatrice	100.00

- Decompose the relation student_activity:

ENROLLMENT

SID	***Activity***
100	Skiing
100	Golf
150	Swimming
175	Squash
175	Swimming
200	Swimming
200	Golf

ACTIVITY

Activity	Place	Fee
Skiing	Mt. Dora	200
Golf	Orlando Links	50
Swimming	Lake Beatrice	100
Squash	UCF Courts	100

EXAMPLE: AN UPDATE ANOMALY

ENROLLMENT

SID	***Activity***
100	Skiing
100	Golf
150	Swimming
175	Squash
175	Swimming
200	Swimming
200	Golf

ACTIVITY

Activity	Place	Fee
Skiing	Mt. Dora	200
Golf	Orlando Links	50
Swimming	Lake Beatrice	100
Squash	UCF Courts	100

- If SID 100 drops skiing, we still know the cost of skiing.
- If the fee for golf changes, it must be changed in only one place.
- We can add an activity without a student.

The Evils of Redundancy

- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

- *Functional dependency constraints* are utilized to identify schemas with such problems and to suggest refinements.

Reducing the Redundant Values in Tuples

- GUIDELINE 2:
  - **Design a schema that does not suffer from the insertion, deletion and update anomalies.**
  - If there are any anomalies present, then note them so that applications can be made to take them into account.

Reducing Null Values in Tuples

- Many attributes may be grouped together into a flat relation
  - If many of attributes do not apply to all tuples in the relation??
- Reasons for nulls:
  - Attribute not applicable or invalid
  - Attribute value unknown (may exist)
  - Value known to exist, but unavailable

Reducing Null Values in Tuples

- Problems:
  - Space wastage
  - Understanding the meaning of the attributes & specifying join operations at the logical level
  - How to take aggregate operations in account
  - Having same representation for all NULLs compromises the different meanings they may have
- GUIDELINE 3:
  - **Relations should be designed such that their tuples will have as few NULL values as possible**
  - Attributes that are NULL frequently could be placed in separate relations (with the primary key)

Spurious Tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations
- The relations should be designed to satisfy the lossless join condition.
- No spurious tuples should be generated by doing a natural-join of any relations.

Example: Spurious Tuples

- Consider relation

  EMP_PROJ(<u>Eno</u>, <u>Pnumber</u>, Hours, Ename, Pname, Plocation)

- Decomposition of EMP_PROJ into relations

  EMP_LOCS (<u>Ename</u>, <u>Plocation</u>)

  and

  EMP_PROJ1 (<u>Eno</u>, <u>Pnumber</u>, Hours, Pname, Plocation)

Example: Spurious Tuples

EMP_PROJ(Eno, Pnumber, Hours, Ename, Pname, Plocation)

Eno	Pno	Hours	Pname	Ploc	Ename
11	1	3	x	Ank	a
11	2	4	y	Ist	a
12	1	1	x	Ank	b
12	3	10	z	Esk	b

- Decomposition of EMP_PROJ into relations

EMP_LOCS (Ename, Plocation)

EMP_PROJ1 (Eno, Pnumber, Hours, Pname, Plocation)

EName	Ploc
a	Ank
a	Ist
b	Ank
b	Esk

Eno	Pno	Hours	Pname	Ploc
11	1	3	x	Ank
11	2	4	y	Ist
12	1	1	x	Ank
12	3	10	z	Esk

Example: Spurious Tuples

EName	Ploc
a	Ank
a	Ist
b	Ank
b	Esk

Eno	Pno	Hours	Pname	Ploc
11	1	3	x	Ank
11	2	4	y	Ist
12	1	1	x	Ank
12	3	10	z	Esk

Now join these two relations

Eno	Pno	Hours	Pname	Ploc	Ename
11	1	3	x	Ank	a
11	1	3	x	Ank	b
11	2	4	y	Ist	a
12	1	1	x	Ank	a
12	1	1	x	Ank	b
12	3	10	z	Esk	b

Example: Spurious Tuples

- Consider relation

  EMP_PROJ(Eno, Pnumber, Hours, Ename, Pname, Plocation)

- Natural Join of relations results in spurious tuples

  EMP_LOCS (Ename, Plocation)

  and

  EMP_PROJ1 (Eno, Pnumber, Hours, Pname, Plocation)

Problem:

Join attribute Plocation is neither a primary key nor a foreign key in either EMP_LOCS or EMP_PROJ1

Spurious Tuples

- GUIDELINE 4:
  - **Design relation schemas so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys**
    - This guarantees that no spurious tuples are generated
  - Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations
    - Joining on such attributes may produce spurious tuples

Spurious Tuples

- There are two important properties of decompositions:

  a) Non-additive or losslessness of the corresponding join

  b) Preservation of the functional dependencies.

Functional Dependencies

Functional Dependencies

- Functional dependencies (FDs)
  - used to specify *formal measures* of the "goodness" of relational designs
  - **constraints** that are derived from the *meaning* and *interrelationships* of the data attributes

Functional Dependencies

- X -> Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y
  - For any two tuples t1 and t2 in any relation instance r(R): If t1[X]=t2[X], *then* t1[Y]=t2[Y]
- X -> Y in R specifies a *constraint* on all relation instances r(R)
- FDs are derived from the real-world constraints on the attributes

Functional Dependencies

- Given a value for attribute x:

  If there is only one corresponding value for attribute y, then x ***determines*** y.

  – y is said to be ***functionally dependent*** on x.

  – x is called a ***determinant***.

  – y may or may not determine x.

  – Candidate keys are determinants.

  – Determinants are candidate keys for the things they determine.

Example Determinants

- EmployeeID determines EmployeeName

  written as EmployeeID → EmployeeName
- Why does EmployeeName NOT determine EmployeeID?
- ProductID → Product Name
- Everyone in the same dorm pays the same fee, Dorm →Fee

  Do you think Fee would determine Dorm?
- If a student may not repeat a course

  CourseNo. + StudentNumber → Grade
- FacultyID + CourseNo. + Section → Room, Time
- FacultyID + Time → Room, CourseNo., Section

Concatenated Determinants

Functional Dependencies: Example

COMPANY (<u>SSN</u>, <u>PNO</u>, HOURS, BONUS, ENAME, PHONE, PNAME, {PLOCATIONS})

- SSN $\rightarrow$ ENAME, PHONE
- PNO $\rightarrow$ PNAME, {PLOCATIONS}
- HOURS $\rightarrow$ BONUS
- {SSN, PNO} $\rightarrow$ HOURS (not BONUS)

FD constraint

- An FD is a property of the attributes in the schema R
- The constraint must hold on *every* relation instance r(R)
- If K is a key of R, then K functionally determines all attributes in R
  - Since we never have two distinct tuples with t1[K]=t2[K]

Full Functional Dependency

- Attribute y is fully functionally dependent on attribute x, if it is functionally dependent on x and not functionally dependent on any proper subset of x.
- A FD x -> y where removal of any attribute from x means the FD does not hold any more
- Example:

R (SSN, PNO, HOURS, ENAME)

FD: {SSN, PNO} → HOURS (full)

FD: {SSN, PNO} → ENAME (not a full FD)

(a partial dependency as SSN -> ENAME also holds)

Inference Rules for FDs

- Given a set of FDs F, we can **infer** additional FDs that hold whenever the FDs in F hold

- Armstrong's inference rules:
  - IR1. (**Reflexive**) If Y *subset-of* X, then X -> Y
  - IR2. (**Augmentation**) If X -> Y, then XZ -> YZ
    - Notation: XZ stands for X U Z
  - IR3. (**Transitive**) If X -> Y and Y -> Z, then X -> Z

- IR1, IR2, IR3 form a **sound** and **complete** set of inference rules

Inference Rules for FDs

- Some additional and useful inference rules:
  - IR4. **(Decomposition)**

    If X -> YZ, then X -> Y and X -> Z
  - IR5. **(Union)** If X -> Y and X -> Z, then X -> YZ
  - IR6. **(Psuedotransitivity)**

    If X -> Y and WY -> Z, then WX -> Z

- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

Inference Rules: Example

- Given F = {A->B, C->X, BX->Z}, derive AC->Z

  A->B : AX->BX (Augmentation)

  AX->BX and BX->Z : AX->Z (transitivity)

  C->X : AC->AX (Augmentation)

  AC->AX and AX->Z : AC->Z (transitivity)

- Given F = {A->B, C->D}, with C is subset of B, show that A->D

  B->C (C is subset of B)

  A->B and B->C so A->C (Transitivity)

  A->C, C->D so A->D (Transitivity)

Inference Rules for FDs

- **Closure** of a set F of FDs is the set F^+ of all FDs that can be inferred from F

- **Closure** of a set of attributes X with respect to F is the set X^+ of all attributes that are functionally determined by X

- X^+ can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in F

Problem: Finding FDs

- Approach 1: During Database Design
  - Designer derives them from real-world knowledge of users
  - Problem: knowledge might not be available
- Approach 2: From a Database Instance
  - Analyze given database instance and find all FD's satisfied by that instance
  - Useful if designers don't get enough information from users
  - Problem: FDs might be artificial for the given instance

Problem: Finding FDs

- DB designers first specify the set of FDs, F that can easily be determined from the semantics of the attributes of R.
- Then IR1, IR2, IR3 are used to infer additional FDs that will also hold in R.
- A systematic way:
  - Determine each set of attributes X that appears as a L.H.S. of some FD in F and then determine the set of all attributes that are dependent on X
  - For each set of attribute X, determine X^+ of attributes that are functionally determined by X based on F **(X^+: closure of X under F)**

Closure of a set of Attributes

- Example:

  F = { ENO -> ENAME,

  PNUMBER -> {PNAME, PLOCATION},

  {ENO, PNUMBER} -> HOURS}

- Closure sets wrt F

$\{ENO\}^+ = \{ENO, ENAME\}$

$\{PNUMBER\}^+ = \{PNUMBER, PNAME, PLOCATION\}$

$\{ENO, PNUMBER\}^+ = \{ENO, PNUMBER, ENAME, PNAME, PLOCATION, HOURS\}$

Equivalence of Sets of FDs

- Two sets of FDs F and G are **equivalent** if:
  - Every FD in F can be inferred from G, and
  - Every FD in G can be inferred from F
  - Hence, F and G are equivalent if $F^+ = G^+$
- Definition (**Covers**):
  - F **covers** G if every FD in G can be inferred from F
    - (i.e., if G^+ *subset-of* F^+)
- F and G are equivalent if F covers G and G covers F
- Algorithm for checking equivalence of sets of FDs

Minimal Sets of FDs

- Minimal set of dependencies
- Set of dependencies in a standard or canonical form with no redundencies

Minimal Sets of FDs

- A set of FDs F is **minimal** if it satisfies the following conditions:
  1. Every dependency in F has a single attribute for its RHS. **(every dependency is in canonical form)**
  2. We cannot replace any dependency X -> A in F with a dependency Y -> A, where Y proper-subset-of X and still have a set of dependencies that is equivalent to F. **(F is left reduced)**
  3. We cannot remove any dependency from F and have a set of dependencies that is equivalent to F. **(F is non redundant)**

Minimal Sets of FDs

- Every set of FDs has an equivalent minimal set (called Minimal Cover)
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs G that is equivalent to a set F of FDs
- To synthesize a set of relations, we assume that we start with a set of dependencies that is a minimal set

Computing Minimal Sets of FDs

Given set of FDs be *E* : {*B* -> *A*, *D* -> *A*, *AB* -> *D* }.

Find the minimum cover of *E*.

- All above dependencies are in canonical form; (step 1 completed)
- In step 2, determine, if *AB* -> *D* has any redundant attribute on the left-hand side; that is, can it be replaced by *B* -> *D* or *A* -> *D*?
- Since B -> A, by augmenting with *B* on both sides (IR2), we have *BB* -> *AB*, or *B* -> *AB* (i). However, *AB* -> D as given (ii).
- By the transitive rule (IR3), we get from (i) and (ii), *B* → *D*. Hence *AB* -> *D* may be replaced by *B* → *D*.
- A set equivalent to original *E*, say *E′* : {*B* -> *A*, *D* -> *A*, *B* -> *D* }.

  No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side.
- In step 3, we look for a redundant FD in E′. By using the transitive rule on *B* → *D* and *D* → *A*, we derive *B* → *A*. Hence *B* → *A* is redundant in E′ and can be eliminated.
- Hence the minimum cover of E is {*B* → *D*, *D* → *A* }.

Exercise

- Consider a relation R (A, B, C, D, E) with FDs:
  AB -> C, B -> D, and C -> E.
  What is/are the key(s) for R?

- Consider a relation R(A,B,C,D,E) with functional dependencies: $A \rightarrow B$, $BC \rightarrow E$, and $D \rightarrow A$
  Find the key for R.

- Consider a relational schema R(A, B, C, D, E) with FDs $AB \rightarrow C$; $D \rightarrow A$, $C \rightarrow D$, $C \rightarrow E$, $E \rightarrow B$ Find all the (minimal) keys for R.