Functional Dependencies and Normalization

Content

Informal Design Guidelines for Relational Databases

- What is relational database design?

  The grouping of attributes to form "good" relation schemas

- Two levels of relation schemas
  - The logical "user view" level
  - The storage "base relation" level
- Design is concerned mainly with base relations
- What are the criteria for "good" base relations?

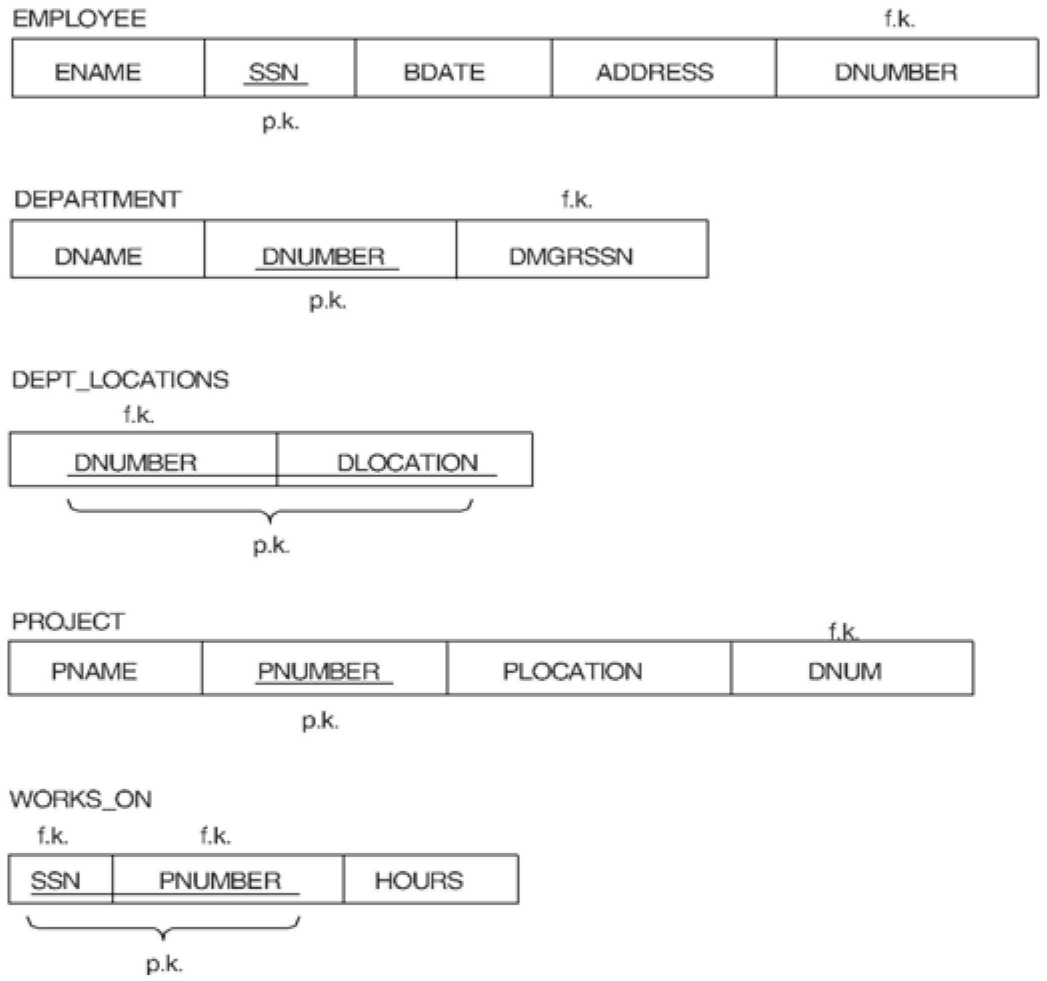
Informal Design Guidelines for Relational Databases

- We first discuss informal guidelines for good relational design
- Then we discuss formal concepts of functional dependencies and normal forms
  - 1NF (First Normal Form)
  - 2NF (Second Normal Form)
  - 3NF (Third Normal Form)
  - BCNF (Boyce-Codd Normal Form)

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, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
- Only foreign keys should be used to refer to other entities
- Entity and relationship attributes should be kept apart as much as possible.



Simplified version of the COMPANY relational database schema.

Redundant Information in Tuples and Update Anomalies

- Mixing attributes of multiple entities may cause problems
- Information is stored redundantly wasting storage
- Problems with update anomalies
  - Insertion anomalies
  - Deletion anomalies
  - Modification anomalies

EXAMPLE OF AN UPDATE ANOMALY (1)

Consider the relation:

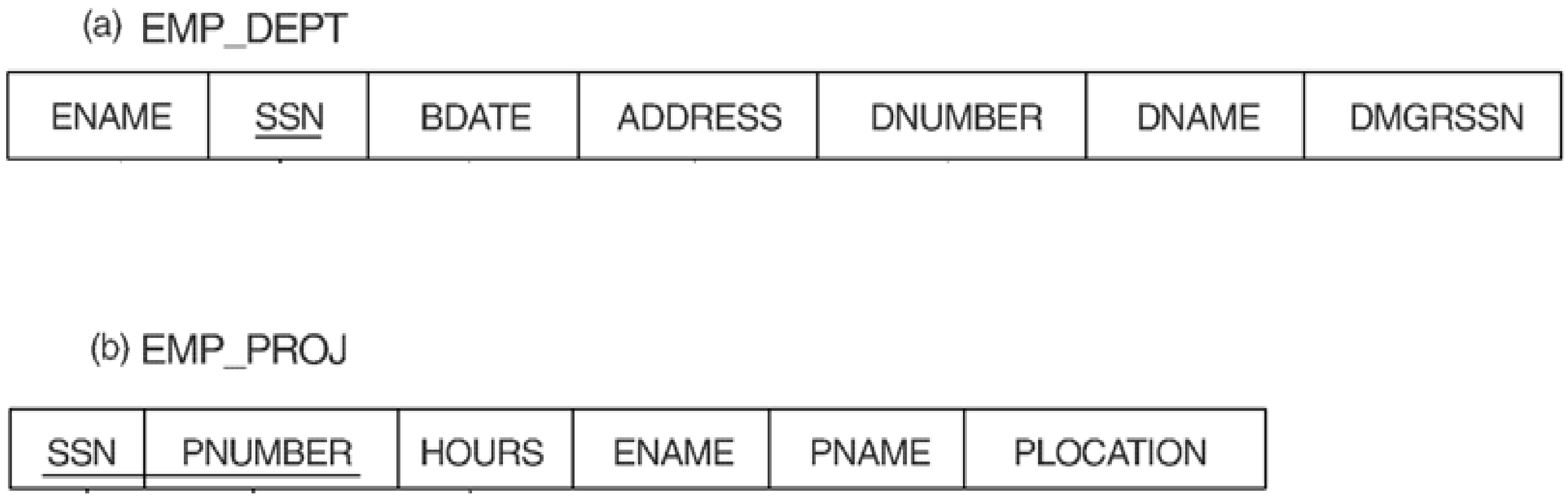
EMP_PROJ (<u>Emp#, Proj#,</u> Ename, Pname, No_hours)

- **Update Anomaly:** Changing the name of project number P1 from "Billing" to "Customer-Accounting" may cause this update to be made for all 100 employees working on project P1.

EXAMPLE OF AN UPDATE ANOMALY

- **Insert Anomaly:** Cannot insert a project unless an employee is assigned to .

  Inversely - Cannot insert an employee unless he is assigned to a project.
- **Delete Anomaly:** When a project is deleted, it will result in deleting all the employees who work on that project. Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.



Two relation schemas and their functional dependencies. Both suffer from update anomalies. (a) The EMP_DEPT relation schema. (b) The EMP_PROJ relation schema.

Guideline to Redundant Information in Tuples and Update Anomalies

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

Null Values in Tuples

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

GUIDELINE 4: 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.

Spurious Tuples (2)

There are two important properties of decompositions:

(a) non-additive or losslessness of the corresponding join

(b) preservation of the functional dependencies.

Note that :

Property (a) is extremely important and *cannot* be sacrificed.

Property (b) is less stringent and may be sacrificed.

Functional Dependencies (1)

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

Functional Dependencies (2)

- A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y
- 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]

Functional Dependencies (2)

- X ➔ Y in R specifies a *constraint* on all relation instances r(R)
- Written as X ➔ Y; can be displayed graphically on a relation schema as in Figures. (denoted by the arrow:).
- FDs are derived from the real-world constraints on the attributes

Functional Dependencies (2)

- 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])

Examples of FD constraints

- social security number determines employee name
  SSN ➔ ENAME
- project number determines project name and location
  PNUMBER ➔ {PNAME, PLOCATION}
- employee ssn and project number determines the hours per week that the employee works on the project
  {SSN, PNUMBER} ➔ HOURS

Inference Rules for FDs (1)

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

<u>**Armstrong's inference rules:**</u>

IR1. (**Reflexive**) If Y *<u>subset-of</u>* X, then X ➔ Y

IR2. (**Augmentation**) If X ➔ Y, then XZ ➔ YZ

(Notation: XZ stands for X ∪ 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 (2)

Some **additional inference rules** that are useful (deduced from IR1, IR2, and IR3):

(**Decomposition**) If X ➔ YZ, then X ➔ Y and X ➔ Z

(**Union**) If X ➔ Y and X ➔ Z, then X ➔ YZ

(**Psuedotransitivity**)

If X ➔ Y and WY ➔ Z, then WX ➔ Z

Inference Rules for FDs (3)

- **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

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 $F^+ = G^+$

<u>Definition:</u> F **covers** G if every FD in G can be inferred from F (i.e., if G^+ <u>*subset-of*</u> F^+)

- F and G are equivalent if F covers G and G covers F
- There is an algorithm for checking equivalence of sets of FDs

Minimal Sets of FDs (1)

- A set of FDs is **minimal** if it satisfies the following conditions:

(1) Every dependency in F has a single attribute for its RHS.

(2) We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.

(3) We cannot replace any dependency X ➔ A in F with a dependency Y ➔ A, where Y proper-subset-of X (Y subset-of X) and still have a set of dependencies that is equivalent to F.

Minimal Sets of FDs (2)

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs 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