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
- Normalization is concerned mainly with base relations

Criteria for "good" base relations

1.1 Semantics of the Relation Attributes

GUIDELINE 1: Informally, each tuple 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.

Bottom Line: Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.

1.2 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

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.

1.3 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)

- Reasons for nulls:

a. attribute not applicable or invalid
b. attribute value unknown (may exist)
c. value known to exist, but unavailable

1.4 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.

Functional Dependencies

- 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
- 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 in R specifies a *constraint* on all relation instances r(R)
- For any two tuples t_1 and t_2 in any relation instance r(R):

  If $t_1[X]=t_2[X]$, *then* $t_1[Y]=t_2[Y]$
- X -> Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y
- FDs are derived from the real-world constraints on the attributes

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 $t_1[K]=t_2[K]$)

Inference Rules for FDs

Armstrong's inference rules:
A1. (Reflexive) If Y subset-of X, then X -> Y
A2. (Augmentation) If X -> Y, then XZ -> YZ
(Notation: XZ stands for X U Z)
A3. (Transitive) If X -> Y and Y -> Z, then X -> Z

- A1, A2, A3 form a *sound* and *complete* set of inference rules

Some additional inference rules that are useful:
(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
The last three inference rules, as well as any other inference rules, can be deduced from A1, A2, and A3 (completeness property)

CLOSURE

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 A1, A2, A3 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 $F^+=G^+$

- Definition: 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
- There is an algorithm for checking equivalence of sets of FDs

Minimal Sets of FDs

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 and still have a set of dependencies that is equivalent to F.

- 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
- Having a minimal set is important for some relational design algorithms

Normal Forms for Relational Databases

- **Normalization:** Process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations

- **Normal form:** Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form
- 2NF, 3NF, BCNF based on keys and FDs of a relation schema
- 4NF based on keys, MVDs; 5NF based on keys,

-Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation;

The purpose of normalizing data

When we design a database for a relational system, the main objective in developing a logical data model is to create an accurate representation of the data, its relationships and constraints. To achieve this objective, we must identify a suitable set of relations. A technique that we can use to help identify such relations is called normalization. Normalization is a technique for producing a set of relations with desirable properties, given the data requirements of an enterprise. Normalization supports database designers by presenting a series of tests, which can be applied to individual relations so that a relational schema can be normalized to a specific form to prevent the possible occurrence of update anomalies.

FIRST NORMAL FORM

First normal form (1NF) is now considered to be part of the formal definition of a relation in the basic (flat) relational model. It states that:
1. The domain of an attribute must include only atomic (simple, indivisible) values and
2. That the value of any attribute in a tuple must be a single value from the domain of that attribute.

Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF disallows relations within relations or relations as attribute values within tuples. The only attribute values permitted by 1NF are single atomic (or indivisible) values.

Consider the DEPARTMENT relation schema, whose primary key is Dnumber, and suppose that we extend it by including the Dlocations attribute. Assuming each department can have a number of locations. This is not in 1NF because Dlocations is not an atomic attribute

SECOND NORMAL FORM

Second normal form (2NF) is based on the concept of full functional dependency. Functional Dependency: The attribute B is fully functionally dependent on the attribute A if each value of A determines one and only one value of B.

Example: PROJ_NUM, PROJ_NAME In this case, the attribute PROJ_NUM is known as the determinant attribute and the attribute PROJ_NAME is known as the dependent attribute.

Generalized Definition: Attribute A determines attribute B (that is B is functionally dependent on A) if all of the rows in the table that agree in value for attribute A also agree in value for attribute B. Fully functional dependency (composite key) If attribute B is functionally dependent on a composite key A but not on any subset of that composite key, the attribute B is fully functionally dependent on A. Partial Dependency: When there is a functional dependence in which the determinant is only part of the primary key, then there is a partial dependency. For example if (A, B) ◊ (C, D) and B◊ C and (A, B) is the primary key, then the functional dependence B◊ C is a partial dependency. {Ssn, Pnumber} → Hours is a full dependency (neither Ssn → Hours nor Pnumber→Hours holds). However, the dependency {Ssn, Pnumber}→Ename is partial because Ssn→Ename holds.