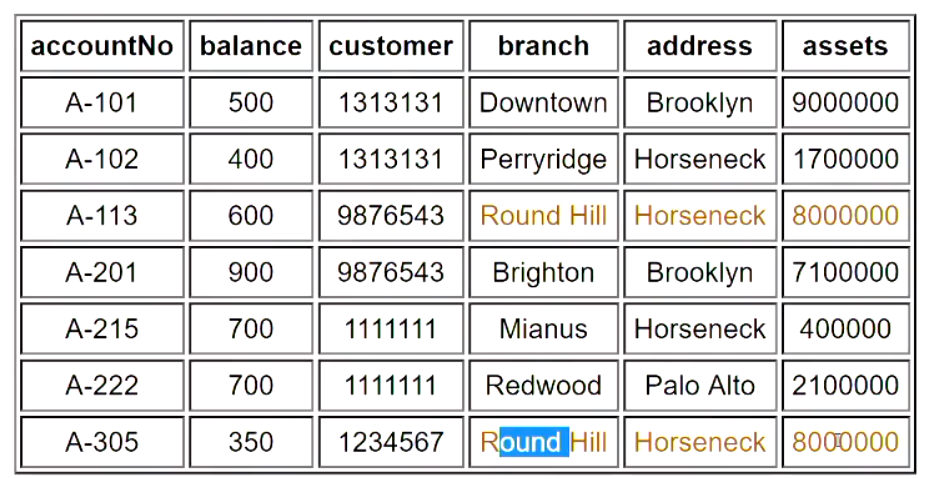
Good relational database design:

* Must capture ALL NECESSARY ATTRIBUTES/ASSOCIATIONS
* Do this with a MINIMAL amount of stored information (no redundant data)



Contain everything in one table will cause redundancy (page 5 example), such as for the same branch with the same address and the same assets, you store them multiple times (same information in multiple tuples) will cause the redundancy

1. Insertion anomaly: when we insert a new record, we need to check that branch data is consistent with existing tuples.

e.g. when you adding the A-305 account in to the table, need to make sure the same branch has the same address and assets.

1. Update anomaly: if a branch changes address, we need to update all tuples referring to that branch
2. **Deletion anomaly: if we you delete the last account of the branch, you will lose the information of the branch.**

(they are avoidable, need to contain the existing branch information and information is up to date)

To avoid these problem:

* Need a schema with MINIMAL OVERLAP between tables
* Each table contains a COHERENT collection (clear and reasonable) of data values (all the info related to the account: account number, account owner should in one table)

1. Start with a universal relation U (a giant table contains all the attributes there)
2. Decompose (divide/break it into smaller part) relation U into several smaller relations Ri (recompose table U to many small tables)
3. Such that each smaller table has minimal overlap with the other smaller table
4. But there is sufficient overlap to reconstruct original table

(usually for each smaller table contains info about one entity e.g. branch, customer…)

|  |  |
| --- | --- |
| Relation schemas | Upper-case letters, denoting set of attributes (R, S, P, Q) |
| Relation instance | Lower-case letter corresponding to schema (r(R), s(S), p(P), q(Q)) |
| tuples | Lower case letters (t, u, v, t’) |
| Attributes | Upper case letters from start of alphabet (A, B, C, D) |
| Set of attributes | Simple concatenation of attribute names X=ABCDE |
| Attributes in tuples | Tuple[attrset] t[ABCD] |

A relation instance *r(R)* satisfies a dependency *X → Y* if

* for any *t, u ∈ r*,   *t[X] = u[X]   ⇒   t[Y] = u[Y]*

In other words, if two tuples in *R* agree in their values for the set of attributes *X*, then they must also agree in their values for the set of attributes *Y*.

We say that "*Y* is functionally dependent on *X*".

Attribute sets *X* and *Y* may overlap; trivially true that *X → X*.

Notes:

* *X → Y* can also be read as "*X* determines *Y*"
* the single arrow *→* denotes functional dependency
* the double arrow *⇒* denotes logical implication

E.g. are there dependencies that hold for *any* relation?

* yes, but they're generally trivial, e.g. *Y ⊂ X   ⇒   X → Y*

E.g. do some dependencies suggest the existence of others?

* yes, rules of inference allow us to derive dependencies
* allows us to reason about sets of functional dependencies

Armstrong's rules are complete, general rules of inference on *fd*s.

|  |
| --- |
| **F1. Reflexivity   e.g.   *X → X***   * **a formal statement of *trivial dependencies*; useful for derivations**   **F2. Augmentation   e.g.   *X → Y  ⇒  XZ → YZ***   * **if a dependency holds, then we can freely expand its left hand side**   **F3. Transitivity   e.g.   *X → Y, Y → Z  ⇒  X → Z***   * **the "most powerful" inference rule; useful in multi-step derivations**   F4. Additivity   e.g.   *X → Y, X → Z   ⇒   X → YZ*   * useful for constructing new right hand sides of *fd*s (also called union)   F5. Projectivity   e.g.   *X → YZ   ⇒   X → Y, X → Z*   * useful for reducing right hand sides of *fd*s (also called decomposition)   F6. Pseudotransitivity   e.g.   *X → Y, YZ → W   ⇒   XZ → W*  shorthand for a common transitivity derivation |

**Normalization:** branch of relational theory providing design insights.

The goals of normalization:

* be able to characterise the level of redundancy in a relational schema
  + can tell how much redundancy we have in this schema design
* provide mechanisms for transforming schemas to remove redundancy

Normalization draws heavily on the theory of functional dependencies.

Normalization theory defines six normal forms (NFs).

* First,Second,Third Normal Forms (1NF,2NF,3NF) (Codd 1972)
* Boyce-Codd Normal Form (BCNF) (1974)
* Fourth Normal Form (4NF) (Zaniolo 1976, Fagin 1977)
* Fifth Normal Form (5NF) (Fagin 1979)

NF hierarachy:   5NF *⇒* 4NF *⇒* BCNF *⇒* 3NF *⇒* 2NF *⇒* 1NF

1NF allows most redundancy;   5NF allows least redundancy.