# Evaluation of DB Design

## The condition of **adequacy**

* The design should allow representing all the important facts about the Database

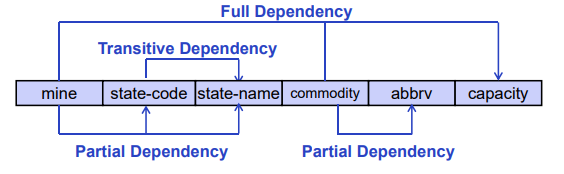
## Condition of **Reduced Redundancy**/ Normalization

* Removal of wrong processes by **removing redundancies** and **creating separate tables**

## Aftereffect of **REDUNDANCY**

|  |  |  |
| --- | --- | --- |
| Insertion anomaly | Deletion Anomaly | Update Anomaly |
| You are forced to create duplicate data or null value when inserting | **Deleting rows** may **delete data permanently** that could be needed in the **future** | **Update** on **one cell** needed update on **many other cells** due to **duplication** |

# Type of dependencies

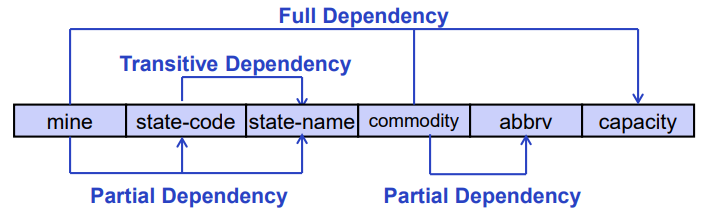
****

|  |  |  |
| --- | --- | --- |
| Partial Dependency | Transitive Dependency | Trivial Dependency |
| When the attribute is determined only by a part of candidate key | When a dependency has MIDDLE MAN  When non-key value determines an attribute | When key pair is used to determine itself |
| Mine -> state-code | Mine -> State-code -> state-name | (Mine, Commodity) -> Mine |

|  |  |
| --- | --- |
| Candidate key | Super Key |
| **Minimal set of attributes** with ability to uniquely identify a row | **Any number of attribute** with ability to uniquely identify a row |
| **Candidate key** can be **Super key**, but **Super key** is not always a **Candidate Key** | |

## Normalization

|  |  |  |
| --- | --- | --- |
| **Type of Normalized form** | **Process** | **Result** |
| **1NF** | Removal of Multivalued attributes |  |
| **2NF** | Removal of Partial Dependencies | All non-key attributes are fully functionally dependent on primary key |
| **3NF** | Removal of Transitive Dependencies | No functional dependencies between Non-primary key attributes |
| **Boyce-Codd normal Form** | Removal of any remaining anomalies resulting from Functional dependencies |  |

****

|  |  |  |
| --- | --- | --- |
| 1NF | 2NF | 3NF |
|  |  |  |

## Table Decomposition

* An act of breaking down table into many components so that
  + Each new **Relation Schema** contains a **subset of the attributes** from **original table**
  + All the dependencies are still present

|  |  |
| --- | --- |
| Lossless Join | Dependency preserving |
| Any form of fact that could be found in the original table should be reachable by the new decomposed table with joins | All functional dependency should hold true even after decompositions |

## Method to find keys from Functional Dependencies

### Attribute closure/ The chase

* + Determining the **close of attributes (X+)**
    - This represents all the attributes accessible by X
    1. Find the first that **cannot be accessed** by any **Attribute**
    2. Then find **another attribute** that **cannot** be accessed by that **initial attribute**
    3. Keep **listing attributes you have access to** and keep finding elements that **you cannot access yet** until you have **all the attributes**
    4. That is the result

### Closure of Functional Dependencies / Armstrong Axioms

* + **Reflexivity rule**
  + **Augmentation rule**
  + **Transitivity rule**

|  |  |  |
| --- | --- | --- |
| Method to find keys from Functional Dependencies | | |
| Attribute closure/ The chase | * + Determining the **close of attributes (X+)**     1. Find the first that **cannot be accessed** by any **Attribute**     2. Then find **another attribute** that **cannot** be accessed by that **initial attribute**     3. Keep **listing attributes you have access to** and keep finding elements that **you cannot access yet** until you have **all the attributes**     4. That is the result | |
| Closure of Functional Dependencies / Armstrong Axioms | Reflexivity Rule |  |
| Augmentation Rule |  |
| Transitivity Rule |  |
| Union Rule |  |
| Decomposition Rule |  |
| Pseudo-Transitivity Rule |  |