**Knowledge and Theory: Relational Databases**

**Part 1 – Attribute Atomicity and 1NF:**

**Relational Database:** A database structured to recognize relations among stored items of information.

**Normalization** is a process of organizing the data in a database to avoid data redundancy and database anomalies (these are bad and we will explain them later). Essentially, it is to organize databases in a way that makes modifying the database easier, quicker and less likely to introduce errors.

**Example**: Look at the table below that displays the top five wing players (shooting guards and small forwards).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| player\_id | player\_name | player\_number | player\_team | player\_endorsement | player\_conference |
| 124 | Kobe Bryant | 8 | Lakers | Nike, Sprite | West |
| 123 | LeBron James | 23 | Cavaliers | Nike | East |
| 101 | Tracy McGrady | 1 | Rockets | Adidas | West |
| 115 | Vince Carter | 15 | Nets | Nike | East |
| 107 | Brandon Roy | 7 | TrailBlazers | McDonald’s | West |

Table 1 – Example Database that fails all Normal Forms

**Attribute Atomicity:** For attributes (or table columns) to be considered atomic, attributes must not be able to be further decomposed **and** can only have one value. For example, “player endorsment” has rows with multiple values (Kobe has two attributes). Attributes must also not be able to be decomposed for it to be atomic. For example, an attribute called “Address” is not atomic since it can be broken down into street name, unit number, street number, etc. This makes someone be able to simultaneously have the addresses “1535 Heatherington Rd” and also “1530 Heathering Rd Unit 190”. The attribute “player name” is not atomic.

**First normal form (1NF)**

A table is in first normal form **if and only if** every attribute in the table is atomic. To make our table atomic, we must split the player name into both first and last name. In additon, we added two rows for “Kobe” to include the two different endorsments he had.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| player\_id | player\_firstname | player\_lastname | player\_number | player\_team | player\_endorsement | player\_conference |
| 124 | Kobe | Bryant | 8 | Lakers | Nike | West |
| 123 | LeBron | James | 23 | Cavaliers | Nike | East |
| 101 | Tracy | McGrady | 1 | Rockets | Adidas | West |
| 115 | Vince | Carter | 15 | Nets | Nike | East |
| 124 | Kobe | Bryant | 8 | Lakers | Sprite | West |
| 107 | Brandon | Roy | 7 | TrailBlazers | McDonald’s | West |

Table 2 – Example Database that passes 1NF but fails 2NF

**Issues with Table 2:**

**Data Redundancy:** To store both of Kobe’s endorsements in this table, you must create two rows for Kobe. Than you must have duplicate information for his unique player ID, team, etc.

**Update anomaly**: If we want to update the jersey number for Kobe (to 24), then we have to update the same thing in multiple rows or the data will become inconsistent.

**Insert anomaly**: Suppose a new employee joins this table who does not have any endorsements. We would not be able to insert the data into the table since most databases don’t accept null.

**Delete anomaly**: Suppose, if at a point of time the NBA bans the endorsements for McDonald’s due to it influencing an unhealthy lifestyle (the same way cigarette advertisements were banned). Then deleting the rows that are having player\_endorsement as McDonald’s would also delete the information of Brandon Roy since he is assigned only to this endorsement.

**2nd Normal Form:** The table must be in first normal form **and** it must have a single row for each primary key attribute. What this means is that you can only have one row for each player. To fix this, we could have two tables. One table would show the player id’s of each player. The other table would show their endorsements. This solves all the issues we had with table two.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| player\_id | player\_firstname | player\_lastname | player\_number | player\_team | player\_conference |
| 124 | Kobe | Bryant | 8 | Lakers | West |
| 123 | LeBron | James | 23 | Cavaliers | East |
| 101 | Tracy | McGrady | 1 | Rockets | West |
| 115 | Vince | Carter | 15 | Nets | East |
| 107 | Brandon | Roy | 7 | TrailBlazers | West |

Table 3A – Example Database that passes 2NF

|  |  |
| --- | --- |
| player\_id | player\_endorsement |
| 124 | Nike |
| 123 | Nike |
| 101 | Adidas |
| 115 | Nike |
| 124 | Sprite |
| 107 | McDonald’s |

Table 3B – Example Database that passes 2NF

This re-organization solves data redundancy, update anomaly, insert anomaly and the delete anomaly.

* We don’t store Kobe’s player number, first and last name, player team, and player conference twice (which solves data redundancy).
* We don’t accidentally forget to update the duplicate information of Kobe (update anomaly). This is because this duplicate information is no longer there in the first place.
* We don’t accidentally delete Brandon Roy when we delete all player endorsements of McDonalds (delete anomaly). This is because we delete all the rows with McDonalds as endorsement from the new table below and it doesn’t affect the table above.
* We don’t insert nulls when we insert a player with no endorsement (insert anomaly). This is because we will simply don’t add a row in the second table.

**Part 2 – Keys and Second Normal Form (2NF):**

**Primary Key** – A Primary Key is a column (or attribute) or a combination of columns (attributes) that uniquely identify a record. There can only be one primary key. For example, we could use the player id attribute to uniquely identify players. Or we could use the player number and player team to uniquely identify a player (since each team ensures that no two players have the same number).

**Candidate Key:** A Candidate Key is any attribute (or column) or a combination of attributes (columns) that can qualify as unique key in database. There can be multiple Candidate Keys in one table. Each Candidate Key can qualify as Primary Key. For example, we already presented “player id” and “player team, player number” as two candidate keys. A candidate key must be “minimal”. This means that it can have a collection of attributes but each attribute must be necessary to uniquely identify the player. For example, “player number, player team, player conference” does uniquely identify a player but it is not minimal. The attribute “conference” was completely unnecessary (it could be deduced by looking at the team) and did not help uniquely identify the player.

**Super Key:** Any attribute or collection of attributes that uniquely identifies a player (or in general, a record). It is essentially a candidate key that does not have to be minimal.

**Prime attribute:** An Attribute which can appear in a candidate key. If you made a list of all the possible candidate keys, any attribute that is in this list is a prime attribute. In this example, player id, player team, player number are all prime attributes. For an intuitive understanding, these attributes are helpful when we are trying to uniquely identify a player (or record). They either let us identify that player or make us one step closer to it.

**Non-prime attribute:** Attribute does not appear in any candidate key. This attribute does not help us identify a player (i.e. player conference or player height).

**Foreign Key:** “Player Id” is a forein key because it is used to connect different tables. For example, you can use it to figure out the endorsement’s of each NBA player in Table 3A and 3B.

**Functional Dependancy:** When one or more columns of a table identify one or more columns of a table. For example, the “player team” identifies the “player conference” attribute. There’s a dependancy there. Any attribute (or collection of attributes) is funcitonally dependant on the candidate keys because the candidate keys identify everything else.

**Note:** Be aware the there are higher levels of normal form such as third normal form, Boyce Codd normal form, etc.

**3rd Normal Form:** The table is in [second normal form](https://en.wikipedia.org/wiki/Second_normal_form), **and** all the attributes in the table are determined only by the [candidate keys](https://en.wikipedia.org/wiki/Candidate_key) of that relation. So for example, the attribute “conference” can be determined by the attribute “team”. ‘Team” is not a candidate key. What this means is that it is completely unnecessary to store the conference of each player. Instead you could store the conference of each team.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| player\_id | player\_firstname | player\_lastname | player\_number | player\_team |
| 124 | Kobe | Bryant | 8 | Lakers |
| 123 | LeBron | James | 23 | Cavaliers |
| 101 | Tracy | McGrady | 1 | Rockets |
| 115 | Vince | Carter | 15 | Nets |
| 107 | Brandon | Roy | 7 | TrailBlazers |

Table 4A – Example Database that passes 3NF

|  |  |
| --- | --- |
| player\_id | player\_endorsement |
| 124 | Nike |
| 123 | Nike |
| 101 | Adidas |
| 115 | Nike |
| 124 | Sprite |
| 107 | McDonald’s |

|  |  |
| --- | --- |
| Team | Conference |
| Lakers | West |
| Cavaliers | East |
| Rockets | West |
| Nets | East |
| TrailBlazers | West |

3NF was designed to improve database processing while minimizing storage costs. Suppose the first table was had two players from the same team. Then you would be storing duplicate information twice. For example, if we added Richard Jefferson of the Nets, then we would be storing the fact that the Nets are in the East twice. Also, we would have to linear search a table (that could consist of 300 NBA Players) just to find what conference a team is in. That wastes time.

**Milestones of Each Normal Form:**

* First Normal form allows you to find each composite attribute. For example, you could find the last name of a particular player without having to interpret the string yourself. You could also separate the two endorsements Kobe instead of getting a string like “Sprite, Nike”. Essentially, it saves you from the task of parsing data entries to correctly separate information in those entries.
* Second Normal Form solves all data redundancies and update anomalies that are still not solved (and can even be introduced) by first normal form.
* 3rd Normal Form saves you some space and time. Note: 3rd normal form saves you time compared to second normal form but it could be slower than a database that is not normalized at all.

**Denormalized vs. Normalized Databases**

Normalized databases are designed to minimize redundancy, remove anamolies, not have to parse data entries and reduce error. Denormalized databases are designed to optimize read time. Suppose you had a database that you knew that you were only going to read from and you did not need to separate composite data.

* The fact that you do not need to separate composite data means there is no benefit to 1NF. If you skip 1NF, you would not introduce data redundancies.
* The fact that you aren’t going to delete, update or insert means there is no benefit to 2NF (since it solves anomalies that aren’t going to happen anyways). In addition, 2NF solves data redundancies which you decide to avoid introducing.
* 3NF only speeds up and saves space for systems in 2NF. It doesn’t speed or save space for a database system that hasn’t been normalized.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| player\_id | player\_name | player\_number | player\_team | player\_endorsement | player\_conference |
| 124 | Kobe Bryant | 8 | Lakers | Nike, Sprite | West |
| 123 | LeBron James | 23 | Cavaliers | Nike | East |
| 101 | Tracy McGrady | 1 | Rockets | Adidas | West |
| 115 | Vince Carter | 15 | Nets | Nike | East |
| 107 | Brandon Roy | 7 | TrailBlazers | McDonald’s | West |

Table 1 – Example Database that fails all Normal Forms

In this case, it is actually faster (to read data) leaving the database system denormalized or denormalizing a database in some normal form. If we wanted to quickly read the conference Brandon Roy plays in (to predict whether he was going to make the All Star game), its faster to do it with a database that is not normalized.