**Normalization (Last Updated 2020-04-21)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Normalization**  Process of designing a *good* database to achieve the following:   * Reduce redundancy; therefore:   + require less disk space   + improve performance * Reduce insertion, deletion, and update anomalies * Reduce the need for restructuring the database as new data attributes are introduced * Reduce data integrity issues * Improve usability of the database | **Example 1:** Project Role Table   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | projId | roleCd | empId | dept | manager | hours | | PX001 | DBA | 22222 | Data | 11111 | 70 | | PX001 | ReqA | 22222 | Data | 11111 | 80 | | PX001 | Test | 33333 | Dev | 10009 | 100 | | PX001 | Developer | 44444 | Dev | 10009 | 300 | | PX001 | Developer | 55555 | Dev | 10009 | 350 | | PX002 | DBA | 22222 | Data | 11111 | 60 | | PX002 | Model | 22222 | Data | 11111 | 60 | | PX002 | Developer | 66666 | Dev2 | 10222 | 250 | | PX002 | Test | 33333 | Dev | 10009 | 80 |   Just by looking at the data, what attributes might functionally define other attributes?  Observations:   * Each occurrence of employee 22222 has the same dept and manager . * Each occurrence of employee 33333 has the same dept and manager . * Each occurrence of dept Data has 11111 as manager . * Each occurrence of dept Dev has 10009 as manager   Given an empId, we know the dept and manager. In other words, empId functionally determines dept and manager .  empId → dept, manager  Given a dept, we know the manager. dept functionally determines manager.  dept → manager  Based on the data, what do you think the primary key is for the Project Role Table?   * Notice the yellow highighted rows. * Notice the pinkish (or maybe light brown) highlighted rows.   Combination of projId, roleCd, empId  projId, roleCd, empId->dept, manager, hours |
| **Anomalies**  What happens if we want to change the manager of a department?   * We must update multiple rows in the Project Role Table. * What if we fail to update one of the rows for the Data dept? If there is supposed to be only one manager per dept, our data would have a data integrity issue.   What if we incorrectly insert a different manager for a dept when inserting a row into the table? | **Example 2**: Data Integrity Issue   |  |  |  |  |  | | --- | --- | --- | --- | --- | | projId | roleCd | empId | dept | manager | | PX001 | DBA | 22222 | Data | 77777 | | PX001 | ReqA | 22222 | Data | 77777 | | PX001 | Test | 33333 | Dev | 10009 | | PX001 | Developer | 44444 | Dev | 10009 | | PX001 | Developer | 55555 | Dev | 10009 | | PX002 | DBA | 22222 | Data | 77777 | | PX002 | Model | 22222 | Data | 11111 | | PX002 | Developer | 66666 | Dev2 | 10222 | | PX002 | Test | 33333 | Dev | 10009 | |
| **First Normal Form (1NF)**  A relation R is in 1NF if all domains of R contain only atomic (non-decomposable) values. | **Example 3: Not 1NF and 1NF versions of a table**  Not in 1NF:   |  |  | | --- | --- | | studentNr | courseNumbers | | 100 | CS1714, MAT1224, HIS1043 | | 200 | MAT1183, HIS1043 |   1NF:   |  |  | | --- | --- | | studentNr | courseNr | | 100 | CS1714 | | 100 | MAT1224 | | 100 | HIS1043 | | 200 | MAT1183 | | 200 | HIS1043 | |
| **1NF is not sufficient**  Notice that the table in example #1 is in 1NF, but it still had issues.  The Enrollment table in example 4 has issues:  1. Insertion   * Cannot insert the fact that professor Jones teaches course CCC001 until the first student enrolls in that course.   2. Deletion   * When the last enrollment row for a particular course is deleted, the prof is lost.   3. Integrity   * If proj Ford replaces professor Jones as the professor for CCC001, all rows with students enrolled in CCC001 must be modified.   4. Redundancy | **Example 4: Problems with an example 1NF relation**  Enrollment(studentNr, courseId, name, prof, office, grade, major)  studentNr → name, major  courseId → prof, office  prof → office  studentNr, courseId → grade |
| **Second Normal Form (2NF)**  A 1NF relation is in 2NF if all non-key attributes of R depend on the entire key.  For the Enrollment relation in example 4:   * name and major are not dependent on the entire key * prof and office are not dependent on the entire key   Note that every relation that is 2NF is also 1NF. | **Example 5: 2NF Relations**  Student (studentNr, name, major)  studentNr → name, major  Course (courseId, prof, office)  courseId → prof, office  prof → office  Enroll(studentNr, courseId, grade)  studentNr, courseId → grade  The example issues 1 thru 3 no longer exist. |
| **2NF is not sufficient**  Redundancy exists in the Course relation since the office is repeated for every course taught by a particular prof. Also, note that an office for a prof cannot be inserted until he teaches a Course.  The problem is the **bad** **transitive functional dependency**.  courseId → prof → office  Let X → Z be a functional dependency. Z is transitively dependent on X if there is an attribute Y such that  X → Y  Y → Z  We may need to remove X → Z, but not all transitive dependencies are *bad*. | A transitive dependency X → Z based on  X → Y → Z  is a bad transitive dependency if both of the following functional dependencies do not exist (i.e., it isn't bad if either of these exists):  Y → X  Z → X  An easier way to determine that it is not bad is if the functional dependencies loop back.  Example 6: Not bad since Y → X    Example 7: Not bad since Z → X |
| **Exercise #1**: does the following relation have a bad transitive dependency?  Employee(empNr, name, salary, SSN)  empNr → name, salary, SSN  SSN → name, salary  SSN → empNr | Answer:  empNr -> SSN -> name, salary  1111 Bob Wire 20,000 111-22-1234  2222 Harry Fingers 30,000 222-22-1111 |
| **Third Normal Form (3NF)**  A 2NF relation is in 3NF if it has no bad transitive functional dependencies. A non-key attribute should not be dependent on a non-key attribute. Note that every 3NF relation is also in 2NF. | **Example 8:Decomposes Course from example 5**  Previously, we had  Course (courseId, prof, office)  courseId → prof, office  prof → office  Decompose Course into  Teaches (courseId, prof)  courseId → prof  Squats (prof, office)  prof → office |
| **Rules for Normalization**  1. Remove bad transitive functional dependencies:   * Given:   + X🡪 Y, Z   + Y 🡪 Z * Remove X 🡪 Z unless it isn't bad because:   + Y 🡪 X, or   + Z 🡪 X   2. If X → Y and X, Z → Y, then X → Y is a stronger FD. Remove the weaker dependency X, Z → Y.  3. If X → Y and X, Y → Z, then reduce X, Y → Z to the stronger X → Z. (Also, keep X → Y ). | **Example #9: Normalize the following relation and identify keys**  ProjectRole(projId, role, empId, dept, manager, hours)  projId, roleCd, empId → dept, manager, hours  empId → dept, manager  dept → manager  **Answer:**  1. Since empId->dept, manager and projId,roleCd,empId->dept,manager, based on rule #2:  Remove projId,roleCd,empId->dept,manager  2. empId -> dept -> manager is a bad transitive dependency so remove empId -> manager.  Project(projId, role, empId, hours)  projId, roleCd, empId → hours  Employee(empId, dept)  empId → dept // given an empId there is a dept  Manages(dept, manager)  dept → manager |
| **Exercise #2: Normalize the following relation and identify keys**  Assume this exercise is independent of example #9.  ProjectTeams(empId, projId, projNm, empNm, projLeadEmpId,dept, manager, duty, title)  empId -> empNm  projId -> projNm, projLeadEmpId  projId -> dept, manager  empId, projId -> duty, empNm, projNm  empId, projId -> dept, manager  dept -> manager, title  manager-> dept | Proposed answer:  Employee(empId, empNm)  empId -> empNm  Department(dept, manager, title)  dept -> manager, title  Manages(manager, dept)  manager-> dept  Project(projId, projNm, projLeadEmpId)  projId -> projNm, projLeadEmpId  ProjectDept(projId, dept)  projId -> dept  Work(empId, projId, duty, dept)  empId, projId -> duty, dept  **Issues with what was done in Exercise #2 proposed answer?**   * Why separate Project and ProjectDept? We shouldn’t since they have the same primary key. * What’s wrong with work? There’s no primary key, and department isn’t needed based on rule 2. * Why separate Department and Manages? We should not   Dept - > manager, title  Manager - > Dept  Transitive dependency : manager -> dept -> title  Is bad if 2nd or 3rd objects don’t reference manager.  But dept functionally determines manager so it’s good! |
| **Work for normalizing #9 (copied, now cross out)**  empId -> empNm  projId -> projNm, projLeadEmpId  projId -> dept, manager  empId, projId -> duty, ~~empNm~~, ~~projNm~~  empId, projId -> ~~dept~~, ~~manager~~  dept -> manager, title  manager-> dept  **Reduction Rules**   1. By rule 2: empId -> empNm is stronger than  empId, projId -> empNm, remove the later. 2. By rule 2: projId -> projNm is stronger than  empId, projId -> projNm, remove the later 3. By rule 2: projId -> dept, manager is stronger than  empId, projId -> dept, manager, remove the later 4. By rule 1: projId -> dept -> manager and dept does not determine projId and manager does not determine projId: remove projId -> manager | **Normalized Relations:**  Employee(empId, empNm)  emp -> empNm  Project(projId, projNm, projectLeadEmpId, dept) either dept or manager but not both  projId -> projNm, projLeadEmpId, dept    Work (empId, projId, duty)  empId, projId -> duty  Department(dept,Manager, title) manager could be primary key, but not the combination.  dept -> manager, title  manager-> dept |
| **Example #10: Normalize relation R into 3NF relations without loss of informaiton**  R (A, B, C, D, E, F, G, H, I, J, K)  A, B, C -> D, E, F, G, H, I, J, K  A -> D  A, D -> E  B, C -> G, H, I  F -> J  H -> I  I -> H  **temp FD list to work with:**  R (A, B, C, D, E, F, G, H, I, J, K)  A, B, C -> ~~D~~, ~~E~~, F, ~~G, H, I~~, ~~J~~, K  A -> D  A, ~~D~~ -> E  B, C -> G, H, ~~I~~  F -> J  H -> I  I -> H | **Example #10** **discussion**:  1. By rule 2: A -> D is stronger than A, B, C -> D:  Remove A, B, C -> D  2. By rule 3: A -> D and A, D -> E:  3. If X → Y and X, Y → Z, then reduce X, Y → Z to the stronger X → Z. (Also, keep X → Y ).  A, ~~D~~ -> E  3. By rule 2: A -> E is stronger than A, B, C -> E  Remove A, B, C -> E  4. By rule 2: B, C -> G, H, I is stronger than A,B,C-> G,H,I  Remove A, B, C -> G, H, I  5. By rule 1: A,B,C -> F -> J and F does not determine A,B,C and J does not determine A, B, C:  Remove A, B, C -> J  6. By rule 1: B,C -> H -> I and H does not determine B,C and I does not determine B, C:  Remove B, C -> I  **A,B,C -> F -> J bad transitive dependency**  **B, C -> H -> I Bad transitive dependency** |
|  | For example #10: Resulting relations, keys, and FDs after normalization:  R1 (A, B, C, F, K)  A, B, C -> F, K  R2 (A, D, E)  A -> D, E  R3 (B, C, G, H) Could have used I instead of H. not b  B, C, -> G, H  R4 (F, J)  F -> J  R5 (H, I) either could be the key, but not both  H -> I  I -> H |
| **Summary of 1NF, 2NF, and 3NF**  A, B, C -> D, E, F, G, H, I, J, K = 1NF   |  |  |  | | --- | --- | --- | | **Normal Form** | **Definition** | **Informal** | | 1NF | Each attribute must not be a repeating group. For each non-key attribute, attri, in the relation:  key -> attri | All non-key attributes must be dependent on the key. | | 2NF | All non-key attributes should not be dependent on part of the key. If the key is a composite of multiple attributes, an attribute, attri, can't be dependent on a subordinate key attribute.  2. If X → Y and X, Z → Y, then X → Y is a stronger FD.  Remove the weaker dependency X, Z → Y.  3. If X → Y and X, Y → Z, then reduce X, Y → Z to the stronger X → Z. (Also, keep X → Y ). | All non-key attributes must be dependent on the entire key. | | 3NF | A 2NF relation is in 3NF if it has no bad transitive functional dependencies. If X → Y and Y → Z,  X 🡪 Z is a bad transitive dependency unless either  Y → X  Z → X | All non-key attributes must be dependent on nothing but the key. |   Informal normalization definition for 1NF, 2NF, and 3NF: Every non-key attribute must be dependent on the key, the entire key, and nothing but the key. | |
| **Is 3NF sufficient?**  No, there are relationships between attributes which are not functional dependencies known as **multi-valued dependencies**. **MVDs** are not considered by 3NF. | **Example #11: MVD**  Research(empId, researchTopic)  empId **->>** researchTopic |
|  |  |
| **Symbols used within this document** | Y → X |

**©2020Larry W. Clark,** UTSA CS students may make copies for their personal use