# ­Chapter 3: Design Theory for Relational Databases

## 3.1 Functional Dependencies

|  |
| --- |
| **Definition of Functional Dependencies**  A ***functional dependency*** (FD) on a relation R is a ***statement of the form "If two tuples of R agree on all of the attributes A1, A2, ..., An*** (i.e., the tuples have the same values in their respective components for each of these attributes), ***then they must also agree on all of another list of attributes B1, B2, ..., Bm***. “  We ***write*** this FD formally ***as A1, A2 , ... , An 🡪 B1, B2 , ... , Bm*** and say that "A1, A2 , ... , An functionally determine B1, B2, ..., Bm"  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 1.15.53 PM.png  ***If we can be sure every instance of a relation R will be one in which a given FD is true***, then we say that ***R satisfies the FD***. It is important to remember that when we say that R satisfies an FD f, we are asserting a constraint on R, not just saying something about one particular instance of R. It is ***common for the right side of an FD to be a single attribute.***  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 1.18.48 PM.png |

|  |
| --- |
| **Keys of Relations**  We say a ***set of one or more attributes*** {A1, A2, ..., An} is a ***key for a relation*** R if:  1. Those attributes functionally ***determine all other attributes of the relation***. That is, it is impossible for two distinct tuples of R to agree on all of A1, A2, ..., An  2. ***No proper subset of*** {A1, A2, ..., An} ***functionally determines all other attributes of R***; i.e., a key must be minimal.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 1.23.35 PM.png |
| **Superkeys**  A ***set of attributes that contains a key*** is called a ***superkey***, short for "superset of a key." Thus, ***every key is a superkey***.  However, some ***superkeys are not (minimal) keys***.  Note that every super key satisfies the first condition of a key: ***it functionally determines all other attributes of the relation***. However, a super key need not satisfy the second condition: ***minimality***.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 1.26.40 PM.png |

## 3.2 Rule about Functional Dependencies

|  |
| --- |
| **Reasoning about Functional Dependencies**  Let the tuples agreeing on attribute A be (a1, b1, c1) and (a2, b2, c2).  Since R satisfies A🡪B, and these tuples agree on A, they must also agree on B.  That is, b1 = b2, and the tuples are really (a, b, c1) and (a, b, c2), where b is both b1 and b2. Similarly, since R satisfies B🡪C, and the tuples agree on B, they agree on C.  Thus, c1 = c2 ; i.e., the tuples do agree on C. We have proved that any two tuples of R that agree on A also agree on C, and that is the FD A🡪C.  FD's often can be presented in several different ways, without changing the set of legal instances of the relation. We say:  • Two sets of FD's S and T are ***equivalent*** if the ***set of relation instances satisfying S is exactly the same as the set of relation instances satisfying T***.  • More generally, a set of FD's S follows from a set of FD's T if every relation instance that satisfies all the FD's in T also satisfies all the FD's in S.  Note then that two sets of FD's S and T are equivalent if and only if S follows from T, and T follows from S. |

|  |
| --- |
| **Splitting/Combining Rule**  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 1.38.23 PM.png  That is, we may ***split attributes on the right side*** so that ***only one attribute appears on the right of each FD***. Likewise, we can replace a collection of FD's having a common left side by a single FD with the same left side and all the right sides combined into one set of attributes. In either event, the new set of FD's is equivalent to the old. The equivalence noted above can be used in two ways.  - We can replace an FD A1, A2, ..., An 🡪 B1, B2, ..., Bm by a set of FD's A1, A2, ..., An 🡪 Bi for i = 1, 2, ... , m. This transformation we call the ***splitting rule***.  - We can replace a set of FD's A1, A2, ..., An 🡪 Bi for i = 1, 2, ... , m by the single FD A1, A2, ..., An 🡪 B1, B2, ..., Bm. We call this transformation the ***combining rule***.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 1.43.22 PM.png  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 1.43.37 PM.png |

|  |
| --- |
| **Trivial Functional Dependencies**  A constraint of any kind on a relation is said to be ***trivial*** ***if it holds for every instance of the relation***, regardless of what other constraints are assumed.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 1.44.44 PM.png  a trivial FD has a right side that is a subset of its left side.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 1.44.54 PM.png  There is an intermediate situation in which some, but not all, of the attributes on the right side of an FD are also on the left. This ***FD is not trivial***, but it ***can be simplified by removing from the right side of an FD those attributes that appear on the left***. That is:  FD A1, A2, ..., An 🡪 B1, B2, ..., Bm is equivalent to A1, A2, ..., An 🡪 C1, C2, ..., Ck  where the C's are all those B's that are not also A's.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 1.45.46 PM.png |

|  |
| --- |
| **Computing the Closure of Attributes**  Before proceeding to other rules, we shall give a general principle from which all true rules follow. Suppose {A1, A2, ..., An} is a set of attributes and S is a set of FD's. The closure of {A1, A2, ..., An} under the FD's in S is the set of attributes B such that every relation that satisfies all the FD's in set S also satisfies A1, A2, ..., An 🡪B.  That is, A1, A2, ..., An 🡪 B follows from the FD's of S. We denote the closure of a set of attributes A1, A2, ..., An by { A1, A2, ..., An }+.  Note that A1, A2, ..., An are always in {A1, A2, ..., An }+ because the FD A1, A2, ..., An 🡪 Ai is trivial when i is one of 1, 2, ... , n.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.00.51 PM.png  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.00.57 PM.png  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.01.27 PM.png  ***Closure works because***: it claims true FD’s and it discovers all true FD’s. |

|  |
| --- |
| **Transitive Rule**  The transitive rule lets us cascade two FD's, and generalizes the observation of Example 3.4.  • If A1, A2, ..., An 🡪 B1, B2, ..., Bm and B1, B2, ..., Bm 🡪 C1, C2, ..., Ck hold in relation R, then A1, A2, ..., An 🡪 C1, C2, ..., Ck also holds in R.  If some of the C's are among the A's, we may eliminate them from the right side by the trivial-dependencies rule.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.05.35 PM.png |
| **Closing Sets of Functional Dependencies**  To avoid some of the explosion of possible bases, we shall limit ourselves to considering only bases whose FD's have singleton right sides. If we have any basis, we can apply the splitting rule to make the right sides be singletons. ***A minimal basis for a relation is a basis B that satisfies three conditions***:  1. All the FD's in B have singleton right sides.  2. If any FD is removed from B, the result is no longer a basis.  3. If for any FD in B we remove one or more attributes from the left side of F, the result is no longer a basis.  Notice that no trivial FD can be in a minimal basis, because it could be removed by rule (2).  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.07.57 PM.png  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.08.02 PM.png |

|  |
| --- |
| **Projecting Functional Dependencies**  Suppose we have a relation R with set of FD's S, and we project R by computing R1=πL(R), for some list of attributes R. What FD's hold in R1?  The answer is obtained in principle by ***computing the projection of functional dependencies S, which is all FD's that***:  a) Follow from S, and  b) Involve only attributes of R1  Since there may be a large number of such FD's, and many of them may be redundant (i.e., they follow from other such FD's), we are free to simplify that set of FD's if we wish.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.10.29 PM.png |

## 3.3 Design of Relational Database Schemas

Careless selection of a relational database schema can lead to ***redundancy and related anomalies***. The ***repetition of this information is redundant***. It also ***introduces the potential for several kinds of errors***. In this section, we shall tackle the ***problem of design of good relation schemas in the following stages***:

1. We first explore in more detail the problems that arise when our schema is poorly designed.

2. Then, we introduce the idea of "decomposition," breaking a relation schema (set of attributes) into two smaller schemas.

3. Next, we introduce "Boyce-Codd normal form," or "BCNF," a condition on a relation schema that eliminates these problems.

4. These points are tied together when we explain how to assure the BCNF condition by decomposing relation schemas.

|  |
| --- |
| **Anomalies**  Problems such as redundancy that occur when we try to cram too much into a single relation are called anomalies. The principal kinds of anomalies that we encounter are:  1. Redundancy  Information may be repeated unnecessarily in several tuples.  2. Update Anomalies  We may change information in one tuple but leave the same information unchanged in another.  3. Deletion Anomalies  If a set of values becomes empty, we may lose other information as a side effect. |

|  |
| --- |
| **Decomposing Relations**  The accepted way to eliminate these anomalies is to decompose relations. Decomposition of R involves splitting the attributes of R to make the schemas of two new relations. After describing the decomposition process, we shall show how to pick a decomposition that eliminates anomalies.  Given a relation R(A1,A2,...,An), we may decompose R into two relations S(B1,B2,..,Bm) and T(C1,C2,...,Ck) such that:  1. {A1, A2, ..., An} = {B1, B2, ..., Bm} ∪ { C1, C2, ..., Ck }.  2. S = πB1, B2, ..., Bm (R).  3. T = π C1, C2, ..., Ck (R).  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.20.09 PM.png  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.20.13 PM.png |

|  |
| --- |
| **Boyce-Codd Normal Form**  The goal of decomposition is to ***replace a relation by several that do not exhibit anomalies***. There is, it turns out, a simple condition under which the anomalies discussed above can be guaranteed not to exist.  This condition is called Boyce Codd normal form, or BCNF.  • A relation R is in BCNF if and only if: whenever there is a nontrivial FD A1,A2,...,An 🡪 B1, B2, ..., Bm for R, it is the case that { A1,A2,...,An } is a ***superkey for R***.  That is, the left side of every nontrivial FD must be a superkey. Recall that a superkey need not be minimal.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.22.20 PM.png |

|  |
| --- |
| **Decomposition into BCNF**  By repeatedly choosing suitable decompositions, we can break any relation schema into a collection of subsets of its attributes with the following important properties:  1. These ***subsets are the schemas of relations in BCNF***.  2. The data in original relation is represented faithfully by the data in the relations that are the result of the decomposition. ***We need to be able to reconstruct the original relation instance exactly from the decomposed relation instances***.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.24.46 PM.png  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.24.33 PM.png |

## 3.4 Decomposition: The Good, The Bad, The Ugly

We shall consider ***three distinct properties*** we would like a decomposition to have.

1. ***Elimination of Anomalies*** by decomposition as in Section 3.3.

2. ***Recoverability of Information***. Can we recover the original relation from the tuples in its decomposition?

3. ***Preservation of Dependencies***. If we check the projected FD's in the relations of the decomposition, can we can be sure that when we reconstruct the original relation from the decomposition by joining, the result will satisfy the original FD's?

It turns out that the BCNF decomposition of Algorithm 3.20 gives us (1) and (2), but does not necessarily give us all three.

|  |
| --- |
| **Recovering Information from a Decomposition**  *Why not just take any relation R and decompose it into relations, each of whose schemas is a pair of R's attributes*? The answer is that the ***data in the decomposed relations***, even if their tuples were each the projection of a relation instance of R, ***might not allow us to join the relations of the decomposition and get the instance of R back***. If we do get R back, then we say the decomposition has a ***lossless join***. |

|  |
| --- |
| **The Chase Test for Lossless Join**  In Section 3.4.1 we argued why a particular decomposition, that of R(A, B, C) into {A,B} and {B,C:}, with a particular FD, B 🡪C, had a lossless join.  Is it true that πS1(R) ⋈ πS2(R) ⋈ … ⋈ πSK(R) = R? Three important things to remember are:   * The ***natural join is associative and commutative***. It does not matter in what order we join the projections; we shall get the same relation as a result. * Any ***tuple t in R is surely*** πS1(R) ⋈ πS2(R) ⋈ … ⋈ πSK(R). The reason is that the projection of t onto Si is surely in πSi(R) for each i, and therefore by our first point above, t is in the result of the join. * As a consequence, πS1(R) ⋈ πS2(R) ⋈ … ⋈ πSK(R) = R when the FD's in F hold for R ***if and only if every tuple in the join is also in R***. That is, the membership test is all we need to verify that the decomposition has a lossless join.   The ***chase test*** for a lossless join is just an organized way to see whether a tuple t in πS1(R) ⋈ πS2(R) ⋈ … ⋈ πSK(R) can be proved, using the FD's in F, also to be a tuple in R.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.35.26 PM.png  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.36.11 PM.png  Remember that our goal is to use the given set of FD's F to prove that t is really in R. In order to do so, we "chase" the tableau by applying the FD's in F to equate symbols in the tableau whenever we can. If we discover that one of the rows is actually the same as t (that is, the row becomes all unsubscripted symbols), then we have proved that any tuple t in the join of the projections was actually a tuple of R.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.37.20 PM.png  At this point, we see that the last row has become equal to t, that is, (a, b, c, d). We have proved that if R satisfies the FD's A🡪B, B🡪C, and CD🡪A, then whenever we project onto {A,D}, {A,C}, and {B,C,D} and rejoin, what we get must have been in R. In particular, what we get is the same as the tuple of R that we projected onto {B, C, D}. |

|  |
| --- |
| **Why the Chase Works**  There are two issues to address:  1. When the chase results in a row that matches the tuple t (i.e., the tableau is shown to have a row with all unsubscripted variables), ***why must the join be lossless***?  2. When, after applying FD's whenever we can, we still find no row of all unsubscripted variables, ***why must the join not be lossless***?  Question (1) is easy to answer. The ***chase process itself is a proof that one of the projected tuples from R must in fact be the tuple t that is produced by the join***. We also know that every tuple in R is sure to come back if we project and join. Thus, the chase has proved that the result of projection and join is exactly R.  For Question (2), suppose that we eventually derive a tableau without an unsubscripted row, and that this tableau does not allow us to apply any of the FD 's to equate any symbols. Then ***think of the tableau as an instance of the*** ***relation R. It obviously satisfies*** ***the given FD's, because none can be applied to equate symbols***. We know that the ith row has unsubscripted symbols in the attributes of Si, the ith relation of the decomposition. Thus, when we project this relation onto the Si’s and take the natural join, we get the tuple with all unsubscripted variables. This tuple is not in R, so we conclude that the join is not lossless. |

|  |
| --- |
| **Dependency Preservation**  We mentioned that it is not possible, in some cases, ***to decompose a relation into BCNF relations that have both the lossless-join and dependency-preservation properties***.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.41.08 PM.png |

## 3.5 Third Normal Form

The solution to the problem illustrated by Example 3.25 is to relax our BCNF requirement slightly, in order to allow the occasional relation schema that cannot be decomposed into BCNF relations without our losing the ability to check the FD's. This relaxed condition is called "***third normal form***."

|  |
| --- |
| **Definition of Third Normal Form**  A relation R is in third normal form (3NF) if:  • Whenever A1,A2,...,An 🡪 B1,B2,...,Bm is a nontrivial FD, either { A1,A2,...,An } is a super key, or those of B1,B2,...,Bm that are not among the A's, are each a member of some key (not necessarily the same key).  An attribute that is a member of some key is often said to be prime. |

|  |
| --- |
| **The Synthesis Algorithm for 3NF Schemas**  We can now explain and justify how we decompose a relation R into a set of relations such that:  a) The relations of the decomposition are all in 3NF.  b) The decomposition has a lossless join.  c) The decomposition has the dependency-preservation property.  Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2018-02-19 at 2.45.46 PM.png |

|  |
| --- |
| **Why the 3NF Synthesis Algorithm Works?**  We need to show three things: that the lossless-join and dependency-preservation properties hold, and that all the relations of the decomposition are in 3NF.  1. ***Lossless Join.***  Start with a relation of the decomposition whose set of attributes K is a superkey. Consider the sequence of FD's that are used in Algorithm 3. 7 to expand K to become K+. Since K is a super key, we know K+ is all the attributes. The same sequence of FD applications on the tableau cause the subscripted symbols in the row corresponding to K to be equated to unsubscripted symbols in the same order as the attributes were added to the closure. Thus, the chase test concludes that the decomposition is lossless.  2. ***Dependency Preservation***.  Each FD of the minimal basis has all its attributes in some relation of the decomposition. Thus, each dependency can be checked in the decomposed relations.  3. ***Third Normal Form***.  If we have to add a relation whose schema is a key, then this relation is surely in 3NF. The reason is that all attributes of this relation are prime, and thus no violation of 3NF could be present in this relation. For the relations whose schemas are derived from the FD's of a minimal basis, the proof that they are in 3NF is beyond the scope of this book. The argument involves showing that a 3NF violation implies that the basis is not minimal. |