http://www.ict.griffith.edu.au/~jw/normalization/ind.php#editAttibutes

Normalization

Insertion anamoly: Trying to insert an employee who works in a department with all the information about the department but the information about the department must match the information for other tuples with the same departments

Also, if we try to insert another department but we have no employee, there is an issue.

Deletion: Trying to delete a department, means you have to delete the entire row

Update: If we change the manager for department 5 for one tuple, we must do it for all the tuples with department 5

Sometimes we have a fat relation which is a relation with many attributes and these attributes may contain null values. Storage waste for example visa status (not all students are foreigners) or dateofbirth may not be known.

Superkey Any number of columns that give you a unique tuple versus

Candidate key The least number columns that give you a unique tuple. One of them is a primary key, the other ones are **alternate or secondary** keys

Surrogate versus Natural key: It is all about the primary key. It may be the actual stuff that makes up real information or it may be an id that is made up. The user behind the scenes doesn't know the key. It is used for the database itself. The reason why we have surrogate is because the natural key can be composite and that is long. Also, if natural key changes then you have to change it everywhere. Surrogate is numbers, it is easier to search, it can be a little confusing.

Composite key: Making a key out of many columns

Compound key: In a many to many relationship, when we create the association table, the two keys that are brought down from the parent table to create a composite key are called a compound key

Indexes: Non clustered indexes are like the index of a book, it points to the various data kind of like an array of pointers whereas clustered is like a phone book in which the actual data is organized.

Primary keys: x-> R which means that x determines all the attributes in R. Also, it has to be minimal which means that it is part of the superkey but it is the smallest one.

Closure The way to do this is through x closure. Take a dependency, apply armstrong's axioms to derive all the attributes and if we get all of them then we have a candidate key. From one of them, you pick a primary key.

We have F closure and x closure. F closure is all the dependencies implied and x closure is looking at a particular dependency. In order to find all the dependencies in F closure, apply Armstrong's axioms that are both sound and complete. Sound means that whatever you find will be in F closure and complete because you will be able to find all the dependencies in the closure

Armstrong

Reflexivity

Augmentation

Transitivity

Through the above we can come up with additional inference rules

Union, pseudo-transitivity and decomposition

======

canonical covers

A **canonical cover** of a set of functional dependencies F is a simplified set of functional dependencies that has the same closure as the original set F.
A **canonical** cover is also called as irreducible set as it is a reduced version of the given set of functional dependencies and is free from all the unnecessary or extraneous functional dependencies.

When it comes to canonical form, (according to cannon which is the right stuff), you want to reduce redundancy. If your DBMS has to process less, then it will be more efficient. There are two aspects, one is the attributes, the other is actual FDs

For the attributes

1) Singleton's on the RHS by decomposing
2) Reduce LHS to get Singleton
    a. Remove one attribute from RHS
    b. Figure out the closure of the remaining attributes from RHS
    c. If the attribute that was eliminated can be derived, then eliminate the attribute
    d. Move on to other attributes

Reduce Functional dependency

1) Take the LHS of an FD, try to derive the RHS of the same FD using other FDs. If yes, eliminate the FD

The result is only one canonical form. There may be other ones. How do you check If one canonical form is the same as other. Let's say you have one which is F and another which is G. Is F equivalent which is an equal sign with a tilde on top. If they are equal, then they are exactly the same. The ones that are exactly the same, are equal. To find out if an FD in one is covered by another, take the LHS of an unequal FD for example in G and try to derive its right hand side by doing a closure on F. If the RHS can be derived, then F covers G.

2) Do this with all the unequal FDs with both F and G.
3) If the answers is yes on both sides, then they are equivalent or if it is not, may be one just covers the other.

https://www.youtube.com/watch?v=Q8j4lYeVIek&vl=en
https://www.youtube.com/watch?v=IiEAfRoHIH8&index=17&list=PLvndR40H6vCUSkb88kFdmlSzAzsnQIJBF

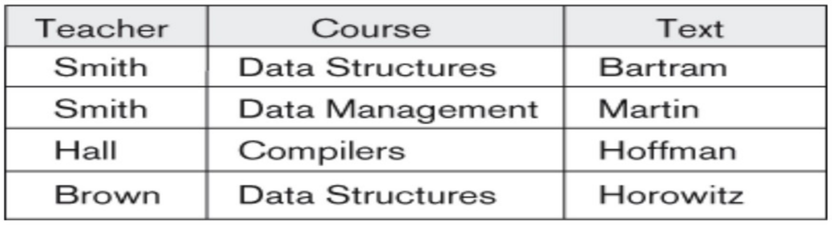
https://www.geeksforgeeks.org/canonical-cover/
https://www.youtube.com/watch?v=IiEAfRoHIH8&list=PLvndR40H6vCUSkb88kFdmlSzAzsnQIJBF&index=17
https://www.youtube.com/watch?v=Zk74s8x0cZ4&list=PLsFENPUZBqiqvQpghfBDZjfcAX0MBuXau&index=22

Functional Dependency

Definition. A functional dependency, denoted by X → Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[Y].

R is relation, r is a relation state or legal extension which is the data in the table at some particular time r(R). The dependencies are not a function of some r but a function of the R. Someone must know the semantics (An understanding of the actual schema) and it cannot be derived just from the r. We just need one counter example to disprove the functional dependency.
Ssn->Ename
Pnumber->{ Pname, Plocation}
{Ssn, Pnumber}-> Hours

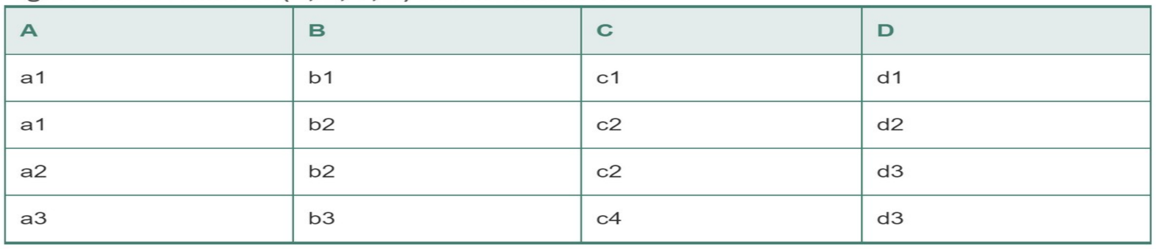
This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; alternatively, the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component. We also say that there is a functional dependency from X to Y, or that Y is functionally dependent on X. The abbreviation for functional dependency is FD or f.d. The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side.
Thus, X functionally determines Y in a relation schema R if, and only if, whenever two tuples of r(R) agree on their X-value, they must necessarily agree on their Y-value. Note the following: If a constraint on R states that there cannot be more than one tuple with a given X-value in any relation instance r(R)—that is, X is a candidate key of R—this implies that X → Y for any subset of attributes Y of R (because the key constraint implies that no two tuples in any legal state r(R) will have the same value of X). If X is a candidate key of R, then X → R. If X → Y in R, this does not say whether or not Y → X in R.
A functional dependency is a property of the semantics or meaning of the attributes. The database designers will use their understanding of the semantics of the attributes of R—that is, how they relate to one another—to specify the functional dependencies that should hold on all relation states (extensions) r of R. Relation extensions r(R) that satisfy the functional dependency constraints are called legal relation states (or legal extensions) of R. Hence, the main use of functional dependencies is to describe further a relation schema R by specifying constraints on its attributes that must hold at all times.



Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

Figure 14.7 A relation state of TEACH with a *possible* functional dependency TEXT → COURSE. However, TEACHER → COURSE, TEXT → TEACHER and COURSE → TEXT are ruled out.

Figure 14.8 A relation *R*(A, B, C, D) with its extension.



A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

Here, the following FDs *may hold* because the four tuples in the current extension have no violation of these constraints: *B* → *C; C* → *B;* {*A, B*} → *C;* {*A, B*} → *D;* and {*C, D*} → *B.* However, the following *do not* hold because we already have violations of them in the given extension*: A* → *B* (tuples 1 and 2 violate this constraint); *B* → *A* (tuples 2 and 3 violate this constraint); *D* → *C* (tuples 3 and 4 violate it).
We denote by F the set of functional dependencies that are specified on relation schema R. Typically, the schema designer specifies the functional dependencies that are semantically obvious; usually, however, numerous other functional dependencies hold in all legal relation instances among sets of attributes that can be derived from and satisfy the dependencies in F. Those other dependencies can be inferred or deduced from the FDs in F.

Normalization

The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to *certify* whether it satisfies a certain **normal form**. The process, which proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary, can thus be considered as *relational design by analysis.* Initially, Codd proposed three normal forms, which he called first, second, and third normal form. A stronger definition of 3NF—called Boyce-Codd normal form (BCNF)—was proposed later by Boyce and Codd. All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation. Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies, respectively.
Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of (1) minimizing redundancy and (2) minimizing the insertion, deletion, and update anomalies. An unsatisfactory relation schema that does not meet the condition for a normal form—the **normal form test**—is decomposed into smaller relation schemas that contain a subset of the attributes and meet the test that was otherwise not met by the original relation.
Normal forms, when considered *in isolation* from other factors, do not guarantee a good database design. It is generally not sufficient to check separately that each relation schema in the database is, say, in BCNF or 3NF. Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:

- The **nonadditive join or lossless join property**, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition
- The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting after decomposition

Definition. Denormalization is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form. Unnormalized is a database that hasn't gone through the normalization process like a spreadsheet of data. This is where denormalizing data takes center stage. While normalized data is optimized for entity level transactions, denormalized data is optimized for answering business questions and driving decision making.

Definition. A **superkey** of a relation schema R = {A1, A2, … , An} is a set of attributes S ⊆ R with the property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[S]. A key K is a superkey with the additional property that removal of any attribute from K will cause K not to be a superkey anymore.
The difference between a key and a superkey is that a key has to be minimal; {Ssn} is a key for EMPLOYEE, whereas {Ssn}, {Ssn, Ename}, {Ssn, Ename, Bdate}, and any set of attributes that includes Ssn are all superkeys.
If a relation schema has more than one key, each is called a **candidate key**. One of the candidate keys is arbitrarily designated to be the **primary key**, and the others are called secondary keys. In a practical relational database, each relation schema must have a primary key. If no candidate key is known for a relation, the entire relation can be treated as a default superkey.
Definition. An attribute of relation schema R is called a **prime attribute** of R if it is a member of some candidate key of R. An attribute is called **nonprime** if it is not a prime attribute—that is, if it is not a member of any candidate key.

First Normal Form

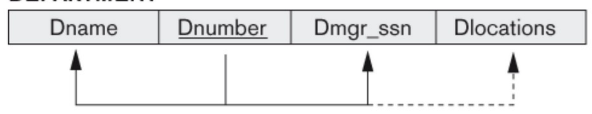
Keep in mind that normal forms cannot be taken in isolation but the process is what makes the good database design.

First normal form (1NF) It states that the domain of an attribute must include only *atomic* (simple, indivisible) *values* and that the value of any attribute in a tuple must be a *single value* from the domain of that attribute. The only attribute values permitted by 1NF are single **atomic** (or **indivisible**) **values**.

Consider the DEPARTMENT relation, whose primary key is Dnumber, and suppose that we extend it by including the Dlocations attribute. We assume that each department can have *a number of* locations.
The DEPARTMENTschema and a sample relation state. As we can see, this is not in 1NF because Dlocations is not an atomic attribute, There are two ways we can look at the Dlocations attribute:

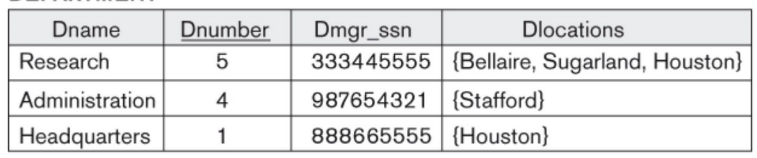
(a)

DEPARTMENT



(b)

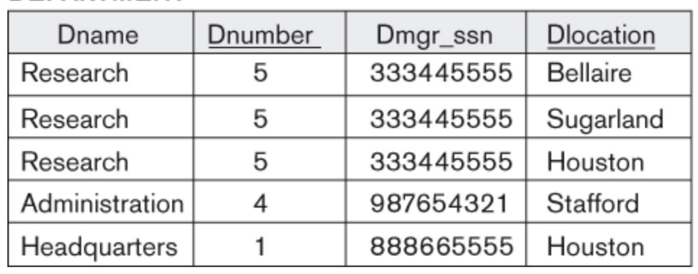
DEPARTMENT



Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT



Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 14.9 Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

CANDIDATE (Ssn, Name, {JOB_HIST (Company, Highest_position, {SAL_HIST (Year, Max_sal)})})
The foregoing describes data about candidates applying for jobs with their job history as a nested relation within which the salary history is stored as a deeper nested relation. The first normalization using internal partial keys Company and Year, respectively, results in the following 1NF relations:
CANDIDATE_1 (Ssn, Name)
CANDIDATE_JOB_HIST (Ssn, Company, Highest_position)
CANDIDATE_SAL_HIST (Ssn, Company, Year, Max-sal)

The existence of more than one multivalued attribute in one relation must be handled carefully. As an example, consider the following non-1NF relation:
PERSON (Ss#, {Car_lic#}, {Phone#})
This relation represents the fact that a person has multiple cars and multiple phones. If strategy 2 above is followed, it results in an all-key relation:
PERSON_IN_1NF (Ss#, Car_lic#, Phone#)
To avoid introducing any extraneous relationship between Car_lic# and Phone#, all possible combinations of values are represented for every Ss#, giving rise to redundancy. The right way to deal with the two multivalued attributes
in PERSON shown previously is to decompose it into two separate relations, using strategy 1 discussed above: P1(Ss#, Car_lic#) and P2(Ss#, Phone#).

Second Normal Form

Definition. A relation schema R is in 2NF if every nonprime attribute A in R is fully 3functionally dependent on the primary key of R. The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains a single attribute, the test need not be applied at all. T

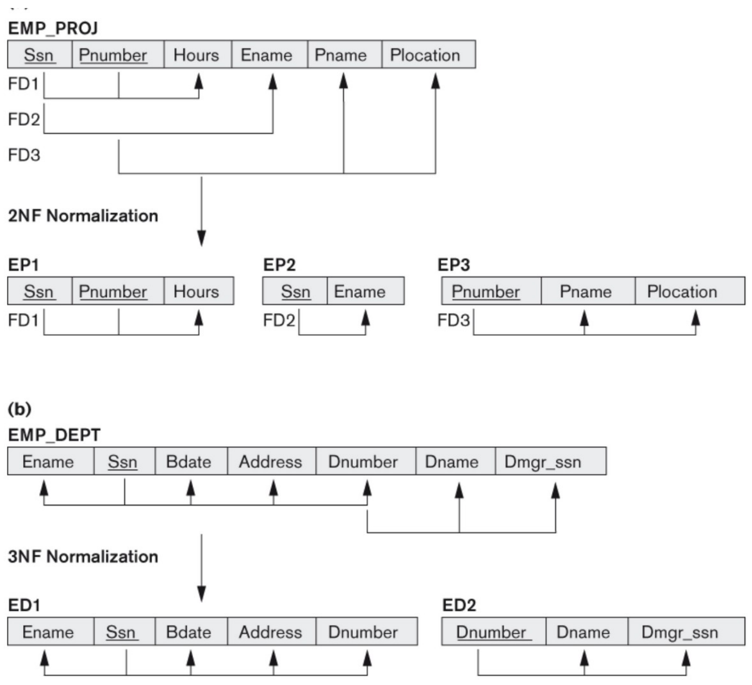


Figure 14.11 Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

Third Normal Form

Third normal form (3NF) is based on the concept of transitive dependency. A functional dependency X → Y in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R.

Definition. According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.

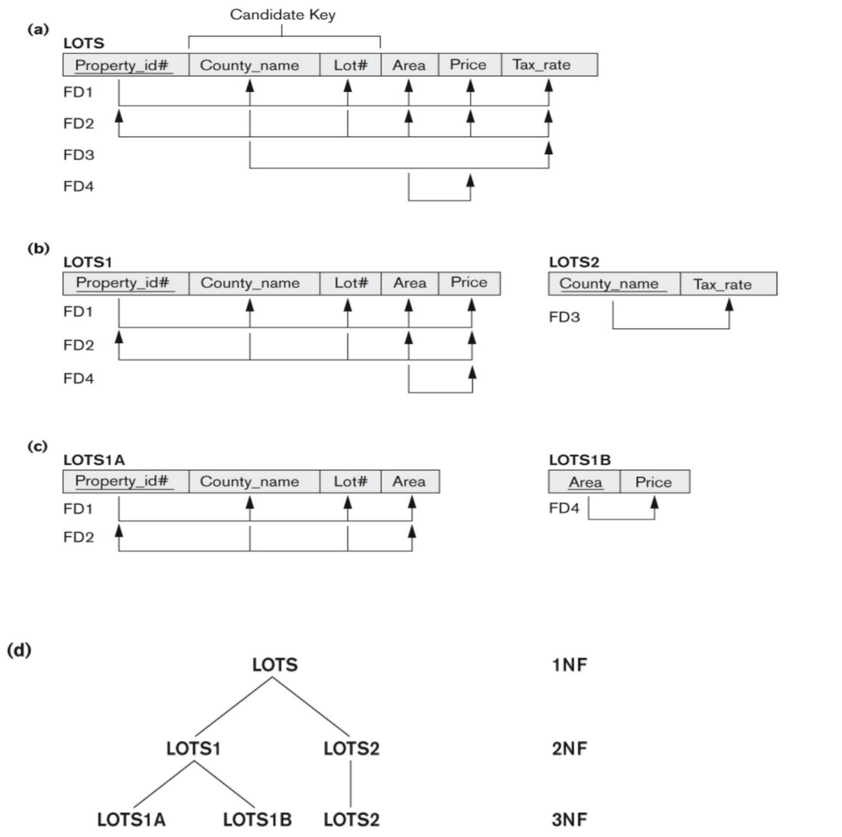


Figure 14.12 Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Progressive normalization of LOTS into a 3NF design.

Definition. A relation schema R is in third normal form (3NF) if, whenever a nontrivial functional dependency X → A holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.

Great Gate questions
https://www.geeksforgeeks.org/functional-dependency-and-attribute-closure/

- **Is the relation in 1NF?** Atomicity, no repeating groups, primary key

- **Is the relation in 2NF?** Is there a composite CK? If not, 2NF is automatically achieved. Otherwise, check your boxes. In *any* of your composite CKs: does a part of it (i.e. a box inside the composite box) by itself determine (point to) a non-prime attribute? If yes, NF2 does not pass (and the relation is thus in 1NF).
- **Is the relation in 3NF?** Does the relation have any non-primes? If no, the relation is automatically in 3NF! Otherwise: Is there a non-prime that determines (point to) another non-prime (resulting in a transitive dependency because the last non-prime is determined indirectly)? If yes, 3NF does not pass (and the relation is thus in 2NF).
- **Is the relation in BCNF?** Are all the determinants also candidate keys? If yes, BCNF pass. This is easiest to check by looking at the dependencies you were given (e.g. A->B and B->{A,C} in example 1) and check that each key (left hand side of the arrow) is in your list of CKs.

1st, 2nd, 3rd and BCNF (only when you get to BCNF) do you get rid of all the insert, update and delete anomalies.

Violation of 1st normal form

No multivalued attributes and no composite attributes.

1) Multivalued means repeating groups
2) Composite attributes Multiple attributes

Violation of 2nd Normal Form

Deals with partial keys. If a proper subset of a key determines a non=prime attribute.

3) Identify the candidate keys (Closure should give you the entire R)
4) Only if we have a composite candidate key do we move forward
5)Identify the prime and the non-prime

When you have a composite key, then do a closure on each single attribute of the prime and see if it contains one nonprime. If it does, then we have violated the 2nd normal form.

Another way to do it is to create two columns when we have a composite key

Prime non prime

Check to find out if a->b if a is prime and b is non-prime. If so, then you have a violation.

To fix it, you have to decompose which is to take the functional dependency that violates and put it in its own table. The remaining goes in another table. These tables have to have a way to connect so you take the LHS of the bad FD and also put it in the good FD

https://www.youtube.com/watch?v=Xslub-nHVss&index=9&list=PLvndR40H6vCUSkb88kFdmlSzAzsnQIJBF

How do you know if you have violated the 3rd normal form

Two ways: Bernstein or the other route

x->a either x is a superkey or a is a prime

It is a superkey if it determines the entire relation
X closure should give you R
If not then check to see if a is a prime which means you have to have the two Categories prime and nonprime and identify what goes into them.

x->y y->z
Y should either be a superkey or z should be a prime otherwise it is a violation. Which means that a nonsuperkey is determining a nonprime.

Find the prime and nonprime (Make sure that there is no violation of 2nd and 3rd normal form)
You apply bernsteins synthesis.
First of all make sure that it is a minimal cover
Then we get rid of the duplicates. Watch video
It preserves all dependencies guaranteed
Take all the subschemas and create different relations
Mix them together.
But it may be lossy.
Take the subschemas and create the grid and see if you can get one row with all the attributes checked, if not, include another subschema with the primary key and then it will be lossless.
https://www.youtube.com/watch?v=TQ3ME70OupU&list=PLvndR40H6vCUSkb88kFdmlSzAzsnQIJBF&index=14

https://www.youtube.com/watch?v=9sADPIE5dQ4

BCNF
Every thing on the LHS of a dependency should be a superkey.
Paired attribute algorithm

Given R and F (R is not in BCNF),

1. Let $X \rightarrow Y$ be a non-trivial FD that holds on R such that $X \rightarrow R$ is not in F^+
   --- X is not a superkey of R and $Y \cap X = \varnothing$
2. Let $R_1 = R - Y$, $R_2 = XY$ (X is a key for R_2)
3. Continue until no BCNF schema is resulted

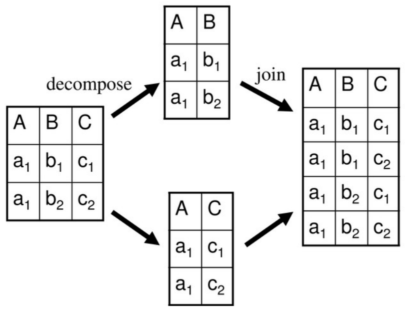
<ex> Lending (B-name, Assets, B-city, Loan#, C-name, Amount)
Key = (Loan# C-name)
F = {B-name→Assets B-city, Loan#→B-name Amount}

1. B-name→Assets, but b-name is not a key
   Decompose into R_1=(B-name, Assets) √
   R_2=(B-name, B-city, Loan#, C-name, Amount)
2. B-name→B-city: R_3=(B-name, B-city) √
   R_4=(B-name, Loan#, C-name, Amount)
3. Loan#→ Amount: R_5=(Loan#, Amount) √
   R_6=(B-name, Loan#, C-name)
4. Loan#→B-name: R_7=(Loan#, B-name) √
   R_8=(Loan#, C-name) √

Find the candidate key (x closure)

Go with three columns L|M|R
Take the functional dependencies and put the attributes in the various columns.
If a column does not appear in any of them, then it must be part of the key
If a column appears on the left hand side, it must also be part of the key even if it is more than one attribute.
If it appears in the middle, it may or may not
If it appears only on the right, then it will not be part of any key
When we say part of any key we are saying that it is either a prime or a non-prime.
Then you would do the closure of what is a prime attribute and add to it as needed and you do the closure.
https://www.youtube.com/watch?v=s1DNVWKeQ_w&index=15&list=PLvndR40H6vCUSkb88kFdmlSzAzsnQIJBF

Lossless and Preserving dependency.



Whenever we do a decomposition we want to not be lossy or lousy but lossless which is always making sure that we are able to get access to everything.

Make a grid with the relations on the left hand side and the attributes on the right hand side.

If we have two relations R1(A) R2(BC) and a set of functional dependency. We put the letter a in each of the fields for each of the relations which are referred to as distinguished variables. Then we use rules to put more distinguished variables in a recursive mode based on 3 rules. Go through each FD

5) 2 or more distinguished variables from the left hand side for multiple rows
6) At least one distinguished variable for the right side
7) Add a distinguished variable using the letter a for the empty spot.

If we get one row completely filled out then it is decomposition.

	A	B	C
R1			
R2			

https://www.youtube.com/watch?v=TykMe1A2u6U

What about preserving dependencies

Look at each relation and take a look at the attributes in the FDs, See if all the FDs are able to be presented in the various relations.
https://www.youtube.com/watch?v=iAarosX8ees

Exercises

Lossless and dependency

1) R = (A, B, C), F = {A → B, B → C) –
   R1 = (A, B), R2 = (B, C)
      Lossless, Preserving
   R1 = (A, B), R2 = (A, C)
      Lossless, Preserving
2) R = {A,B,C,D,E}. F = {A →BC, CD →E, B → D, E →A }
      R1 = {A,B,C}, R2 ={A,D,E}
      R1 = {A,B,C}, R2 ={C,D,E}

3) R = (A, B, C), F = {A→B, B→C, A→C}
   R1(A,B), R2(A,C)

   R = (A, B, C), F = {A→B, B→C, A→C}
   R1(A,B), R2(B,C)

Compute the Canonical cover

1. R = (A, B, C) F = {A → BC, B → C, A → B, AB → C}
   {A→B, ABCD→E, EF→GH, ACDF→EG}
   RHS singletons (Decompose)
   {A→B, ABCD→E, EF→G, EF->H, ACDF→E, ACDF->G}
   LHS
      A**B**CD which is ACD closure=ACDB, got the B, stop
      ACD->E, nothing can be done with any of the other ones

ACDF->E, nothing can be done
ACDF->G, nothing can be done
{A→B, ACD→E, EF→G, EF->H, ACDF→E, ACDF->G}

Reduce FDs

ACDF->E, ACDF closure=ACDFE   eliminate
ACDF->G, ACDF closure=ACDFEG eliminate

Final answer

{A→B, ABCD→E, EF→G, EF->H}

Can check if they are equivalent and they both cover each other. For example
ACDF->EG, ACDF closure = ACDFBEGH

What are the candidate keys

R = (A, B, C, G, H, I)
F = {A → B A → C CG → H CG → I B → H}

Is (AG)+ a candidate key?

1. Is R in 3NF, why? If it is not, decompose it into 3NF
2. Is R in BCNF, why? If it is not, decompose it into BCNF

R = (A, B, C, D) F = {AB→C, AB→D, C→A, D→B}

Decompose R into a set of BCNF relations
R =(A, B, C, D). F = {C→D, C→A, B→C}.

Identify the best normal form that R satisfies.
R =(A, B, C, D). F = {C→D, C→A, B→C}.

https://www.youtube.com/watch?v=FOHicipnG74