Super Keys:

- A **super key** is a set of one or more attributes, which can uniquely identify a row in a table.
- A super key may have additional attributes that are not needed for unique identification.
- **E.g. 1.** Suppose that we have a relation called Students with the attributes id, first_name, last_name and average and {id} is a super key.
  Since {id} is a super key, then the following are also super keys:
  - {id, first_name}
  - {id, last_name}
  - {id, average}
  - {id, first_name, last_name}
  - {id, first_name, average}
  - {id, last_name, average}
  - {id, first_name, last_name, average}

  This is because since id can uniquely identify a row in Students, anything else we add to the set can also uniquely identify a row in Students.
- Another way to define super keys is through closure. The closure of a super key should give back the entire relation.
- **E.g. 2.** Given R(A, B, C) and A → BC. Determine if A is a super key.

  Soln:
  The closure of A, A^+ = {A, B, C}.
  Since the closure of A gives back the entire relation R, it is a super key.
- **E.g. 3.** Given R(A, B, C, D) and
  ABC → D
  AB → CD
  A → BCD

  What are the super key(s) if any exist?

  Soln:
  Since we don't have A on the RHS of any fd, we know that our super key must contain at least A.
  Furthermore, we see that A^+ = {A, B, C, D}. Hence, A is a super key. This means that the following are all super keys:
  {A, B}
  {A, C}
  {A, D}
  {A, B, C}
  {A, B, D}
  {A, C, D}
  {A, B, C, D}

Candidate Keys:

- A **candidate key** is a minimal super key.
  I.e. It is the minimal set of attributes needed to uniquely identify a row in a table.
  In example 1, only {id} is a candidate key.
  In example 3, only {A} is a candidate key.
- Properties of candidate keys:
  - It must contain unique values.
  - It may have multiple attributes.
  - It must not contain null values.

    - It should contain the minimum fields to ensure uniqueness.
    - It should uniquely identify each record in a table.
- A table can have multiple candidate keys.
- All candidate keys are super keys but not all super keys are candidate keys.
- **E.g. 4.** Given R(A, B, C, D) and
  B → ACD
  ACD → B

  List out all the candidate keys, if there are any.

  Soln:
  B^+ = {A, B, C, D}
  $(ACD)^+$ = {A, B, C, D}.
  Hence, both {B} and {A, C, D} are both super keys.
  However, because they are both minimal, they are also candidate keys.
  Note that for ACD, you cannot break it down and still get all the attributes in R.
  A^+, C^+, D^+, $(AC)^+$, $(AD)^+$, $(CD)^+$ do not give you all the relations in R.
  Hence, ACD is minimal.
  So in this case, we have 2 candidate keys for the relation R.
- **E.g. 5.** Given R(A, B, C, D) and
  AB → C
  C → BD
  D → A

  List all the candidate keys, if there are any.

  Soln:
  $(AB)^+$ = {A, B, C, D}
  C^+ = {A, B, C, D}
  D^+ = {A, D}

  In this example, both {A, B} and {C} are candidate keys.
- **E.g. 6.** Given R(A, B, C, D) and
  A → B
  B → C
  C → A

  List all the candidate keys, if there are any.

  Soln:
  First, notice that neither A, B nor C can get you column D. Hence, we know that our candidate key must contain D.

  A^+ = {A, B, C}
  B^+ = {A, B, C}
  C^+ = {A, B, C}

  Hence, the candidate keys are {A, D}, {B, D}, and {C, D}.

- **E.g. 7.** Given R(A, B, C, D) and
  AB → CD
  D → A
  List all the candidate keys, if there are any.

  Soln:
  $(AB)^+$ = {A, B, C, D}
  D^+ = {A, D} ← Notice that the closure of D has all the relations except for B and C. We know that AB gets us B and C and we already have A.
  $(BD)^+$ = {A, B, C, D}

  Hence, {A, B} and {B, D} are candidate keys.
- **E.g. 8.** Given R(A, B, C, D, E, F) and
  AB → C
  C → D
  B → AE

  List all the candidate keys, if there are any.

  Soln:
  $(AB)^+$ = {A, B, C, D, E}
  B^+ = {A, B, C, D, E} ← Only missing column F.
  C^+ = {C, D}

  Hence, {B, F} is the only candidate key.
- **E.g. 9.** Given R(A, B, C, D) and
  AB → CD
  C → A
  D → B

  List all the candidate keys, if there are any.

  Soln:
  $(AB)^+$ = {A, B, C, D}
  C^+ ={A, C} ← Missing B and D. We know that AB gets us CD, so {B, C} is a candidate key as we already have A.
  D^+ = {B, D} ← Missing A and C. We know that AB gets us CD, so {A, D} is a candidate key as we already have B.

  Hence, {A, B}, {B, C}, {C, D} and {A, D} are the candidate keys.

Primary Keys:

- A **primary key** is a chosen candidate key.
  I.e. There could be multiple candidate keys. From the options, we choose one to use. The one that we chose to use is the primary key.
- Rules for defining primary keys:
  - Two rows can't have the same primary key value.
  - The primary key field cannot be null.
  - The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.
- **Prime attributes** are the attributes of the candidate key(s).
- **Non-prime attributes** are the attributes of a table not in the candidate key(s).

Normalization:

- **Normalization** is a database design technique that reduces data redundancy and eliminates undesirable characteristics like insertion, update and deletion anomalies.
- Normalization divides larger tables into smaller tables and links them using relationships.
- The purpose of normalization is to eliminate repetitive data and ensure data is stored logically.
- **E.g. 10.** Consider the table below:

  Student

SID	Name	Program	Department Head	Department Head's Phone Number
1	A	CSC	X	100-100-1000
1	A	MAT	Y	100-100-1001
2	B	CSC	X	100-100-1000
3	C	MAT	Y	100-100-1001
4	D	STA	Z	100-100-1002

  Here are some problems with this design:

  1. Suppose we enroll a new student who's not in any program. Then, the program, department head and department head's phone number will be blank. This is an example of **insertion anomaly**.
  2. Suppose that a department head gets changed. Then, we would have to change that information for multiple students, and if by mistake we miss any record, it will lead to data inconsistency. This is an example of **updation anomaly**.
  3. We see that the department head and department head's phone number information are repeated for the students who are in that program. This is an example of **data redundancy**.
  4. Suppose that student D graduated and all rows pertaining to student D gets deleted. If student D is the only student in the stats program, then we lose important information, such as student D's program, the program's department chair and the department chair's phone number, when we delete all rows pertaining to student D. This is an example of **deletion anomaly**.

- Anomalies are caused when there is too much redundancy in the database's information.
- **Update anomaly** happens when there are multiple entries of the same data in the db and when we update that data, one or more entries do not get updated. Then, we will have data inconsistency.
- **Insertion anomaly** happens when inserting vital data into the database is not possible because other data is not already there.
- **Deletion anomaly** happens when the deletion of unwanted information causes desired information to be deleted as well.
- There are a few normalization rules we can use:
  - 1NF (First Normal Form)
  - 2NF (Second Normal Form)
  - 3NF (Third Normal Form)
  - BCNF (Boyce and Codd Normal Form)
  - 4NF (Fourth Normal Form)

- **1NF (First Normal Form):**
- For a table to be in the First Normal Form, it must follow the following rules:
    1. Each table cell should contain a single value.
    2. Each record needs to be unique.
    3. Values stored in a column should be of the same domain
    4. All the columns in a table should have unique names.
- **2NF (Second Normal Form):**
- For a table to be in the Second Normal Form, it must follow the following rules:
    1. It is already in First Normal Form.
    2. It must not have **partial dependency**. **Partial dependency** occurs when a non-prime attribute in a table depends on only a part of the candidate key and not on the whole candidate key.
       I.e. A partial dependency occurs when we have P → NP where P is 1 or more prime attributes but is not a candidate/primary key and NP is 1 or more non-prime attributes.
       E.g. 11. Consider R(A, B, C, D) and
       AB → D
       B → C

       We see that the candidate key is {A, B}. However, R is not in 2NF because the attribute C only depends on B and not A & B. This is an example of partial dependency.

       To change R to 2NF, we have to decompose it so that the partial dependencies are its own tables.

       For this example, we decompose R(A, B, C, D) into
       R1(A, B, D) and
       R2(B, C)
       Note: When you decompose R into smaller relations, you always want a relation with the primary keys. In this case, we have R1, so we don't need an additional table.
- **E.g. 12.** Consider R(A, B, C) and
  AB → C
  B → C

  We see that the candidate key is {A, B}. We see that B → C is a partial dependency.

  To change R to 2NF, we have to decompose it so that the partial dependencies are its own tables.

  For this example, we decompose R(A, B, C) into
  R1(A, B) and
  R2(B, C)

  We don't have C in R1 because we already have C in R2.
- **E.g. 13.** Consider R(A, B, C, D, E) and
  AB → C
  D → E

We see that the candidate key is {A, B, D}. We see that AB → C and D → E are partial dependencies.

To change R to 2NF, we have to decompose it so that the partial dependencies are its own tables.

For this example, we decompose R(A, B, C, D, E) into
R1(A, B, C)
R2(D, E)
R3(A, B, D)

- **E.g. 14.** Consider R(A, B, C, D, E) and
  A → B
  B → E
  C → D

  We see that the candidate key is {A, C}. We see that A → B and C → D are partial dependencies.

  To change R to 2NF, we have to decompose it so that the partial dependencies are its own tables.

  For this example, we decompose R(A, B, C, D, E) into
  R1(A, B, E)
  R2(C, D)
  R3(A, C)

  Note: B → E is not a partial dependency as B is not part of the primary key. For 2NF, we simply find the relation that contains B, which is R1, and add E to it.
- **3NF (Third Normal Form):**
- A table is in third normal form if:
  1. It is in 2nd normal form.
  2. It must not have **transitive dependencies**.

  Recall: A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies.
  I.e. If A → B and B → C, then A → C is a transitive dependency.
  Another way to think about transitive dependency is that it occurs when a non-prime attribute depends on other non-prime attributes. So, a transitive dependency occurs when you have NP → NP.
- The normalization of 2NF relations to 3NF relations involves the removal of transitive dependencies. If a transitive dependency exists, we remove the transitively dependent attribute(s) from the relation by placing the attribute(s) in a new relation along with a copy of the determinant.
  Recall: The left side of a functional dependency is called the determinant.
- **E.g. 15.** Consider R(A, B, C) and
  A → B
  B → C

  We see that A^+ = {A, B, C}, so {A} is the candidate key and A is a prime attribute. Furthermore, we see that we have a transitive dependency B → C.

What we do is we split R into 2 relations:
R1(A, B)
R2(B, C)

- Let P be prime attribute(s) and NP be non-prime attribute(s) and suppose that {P} is not a candidate/primary key. Then, we have
    1. **Partial dependency** if we have P → NP.
    2. **Transitive dependency** if we have NP → NP.

If we have P/NP → P, we know for sure that it is in 3NF.

- **E.g. 16.** Consider R(A, B, C, D, E) and
A → B
B → E
C → D

We see that $(AC)^+$ = {A, B, C, D, E}, so {A, C} is a candidate key.
We see that we have
1. A → B (Partial dependency)
2. C → D (Partial dependency)
3. B → E (Transitive dependency)

To turn R into 3NF, we will break it down into the following relations:
R1(A, B, E) ← Since B → E, we put E here. However, B → E is a transitive dependency, so we have to split up R1. We will split R1 up into R11 and R12.
R11(A, B)
R12(B, E)
R2(C, D)

R3(A, C) ← **Note:** When you decompose R into smaller relations, you always want a relation with the primary keys. In this case, we need to create a new relation to get a relation with the primary keys.

The final decomposition of R is:
R11(A, B)
R12(B, E)
R2(C, D)
R3(A, C)

- **E.g. 17.** Consider R(A, B, C, D, E, F, G, H, I, J) and
AB → C
A → DE
B → F
F → GH
D → IJ

A candidate key is {A, B} as the closure of (AB) gets back all attributes in R.
We see that
1. A → DE is a partial dependency (pd).
2. B → F is a pd.
3. F → GH is a transitive dependency (td).
4. D → IJ is a td.

We want to decompose R so that it is in 3NF.

We start with R1(A, D, E, I, J). We know that D → IJ, so we put I and J here. However, D → IJ is a td, so we have to split up R1 into R11 and R12.
R11(A, D, E)
R12(D, I, J)
Next, we have R2(B, F, G, H). We know that F → GH, so we put G and H here. However, F → GH is a td, so we split up R2 into R21 and R22.
R21(B, F)
R22(F, G, H)

R3(A, B, C)

The final decomposition of R is:
R11(A, D, E)
R12(D, I, J)
R21(B, F)
R22(F, G, H)
R3(A, B, C)

- **E.g. 18.** Consider R(A, B, C, D, E) and
AB → C
B → D
D → E

A candidate key is {A, B} as the closure of (AB) gets back all attributes in R.
We see that B → D is a pd and that D → E is a td.
We need to decompose R.
We start with R1(B, D, E). We know that D → E, so we put E here. However, D → E is a td, so we have to split R1 into R11 and R12.
R11(B, D)
R12(D, E)

Next, we have R2(A, B, C).

The final decomposition of R is:
R11(B, D)
R12(D, E)
R2(A, B, C)

- **BCNF (Boyce and Codd Normal Form):**
- For a table to be in BCNF, following conditions must be satisfied:
    1. It must be in 3NF.
    2. For each functional dependency (X → Y), X must be a super key.
- **E.g. 19.** Consider R(A, B, C) with the fds
AB → C
C → B

We see that {A, B} and {A, C} are candidate keys.
Hence, A, B and C are all prime attributes.
AB → C is neither a pd nor td.
C → B is neither pd or td because both the LHS and the RHS have prime attributes.
Hence, we see R is in 3NF.
However, R is not in BCNF because in C → B, C is not a super key.

To fix this, I'll decompose R into
R1(C, B)
R2(A, C) ← We chose (A, C) over (A, B) to prevent loss of data when joining R1 and R2.

- **E.g. 20.** Given R(A, B, C, D, E, F, G, H) and
AB → C
A → DE
B → F
F → GH

What form is it?

Soln:
We see that a candidate key is {A, B}.
We see that A → DE is a pd. Hence, R is in 1NF only.

- **E.g. 21.** Given R(A, B, C, D, E) and
CE → D
D → B
C → A

What form is it?

Soln:
We see that a candidate key is {C, E}.
We see that C → A is a pd. Hence, R is in 1NF only.

- **4NF (Fourth Normal Form):**
- For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:
    1. It should be in the **Boyce-Codd Normal Form**.
    2. For each non-trivial multi-valued dependency A -->> B, A is a key.
- A **multi-valued dependency** occurs when two attributes in a table are independent of one another, but both depend on a third attribute.
- Here is the formal definition for multi-valued dependency:
Let R be a relation. Let A, B and rest be attributes. Let t, u, and v be tuples.
$\forall t, u \in R$ if t[A] = u[A], then $\exists v \in R$ s.t. v[A] = t[A] and v[B] = t[B] and v[rest] = u[rest].

We can show a picture of this.
R

	A	B	Rest
t	a	b1	r1
u	a	b2	r2
v	a	b1	r2
w	a	b2	r1

In our definition, we said that "For all tuples t and u in relation R, if t[A] equals to u[A], then there exists a tuple v in R such that v[A] = t[A] and v[B] = t[B] and v[rest] = u[rest]." In the table above, we can see that t[A] = u[A] = a. Furthermore, we didn't specify that t[B] = u[B] or t[rest] = u[rest], so we have different values for those. We can create tuple

v based on the definition and tuples t and u. By swapping the roles of t and u, we can create tuple w.

- A table is said to have multi-valued dependency, if the following conditions are true:
    1. For a dependency A → B, if for a single value of A, multiple values of B exists, then the table may have multi-valued dependency.
    2. A table should have at-least 3 columns for it to have a multi-valued dependency.
    3. For a relation R(A,B,C), if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.
- **Note:** Every FD is an MVD. This is because if X → Y, then swapping Y's between tuples that agree on X does not create new tuples.
  I.e. X → Y implies X -->> Y
- **Note:** If X -->> Y, then X -->> R - Y - X.
- I.e. If we have a relation R(A, B, C, D) and we have A -->> B, then we also have A -->> CD.
- The main idea of 4NF is to eliminate redundancy due to the multiplicative effect of MVDs.
- **E.g. 22.** Consider the table below:

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

We see that course and hobby are independent of each other, but are dependent on stu_id. Furthermore, we see that for stu_id value of 21, there's 2 different corresponding course values and 2 different corresponding hobby values. Hence, the table has multi-valued dependency.

- We use -->> to denote a multi-valued dependency.
  I.e. For the table above in example 22, **stu_id -->> course** and **stu_id -->> hobby**.
- **E.g. 23.** Consider the relation Apply(SSN, collegeName, hobby) and the fact that SSN -->> collegeName (cName) and SSN -->> hobby.

Based on the information above, we can create a table below:

SSN	cName	Hobby
123	Stanford	Trumpet
123	Berkeley	Tennis
123	Stanford	Tennis
123	Berkeley	Trumpet

Notice how all possible combinations of cName and hobby are listed.

- A **trivial multivalued functional dependency** occurs when X -->> Y and

1. $Y \subseteq X$ (Y is a subset of X) or
2. $X \cup Y$ gets back all the attributes of the relation.
   I.e. There's no "rest."

- A **non-trivial multi-valued functional dependency** occurs otherwise.
- Rules of multi-valued functional dependency:
  1. If we have $A \rightarrow B$, then we also have A -->> B.
     Proof:
     Consider the template table below and the fact that $A \rightarrow B$.

	A	B	Rest
t	a1	b1	r1
u	a1	b2	r2

     We want to prove that there exists a tuple v with the following values:

	A	B	C
v	a1	b1	r2

     We know that v exists for the following reasons:
     a. Since $A \rightarrow B$, and we have a1 | b1 and a1 | b2, we know that b1 = b2.
     b. Since b1 = b2, go back to v and rewrite b1 as b2. You'll see that now, you have a1 | b2 | r2. This row exists and is tuple u.
     c. Hence, tuple v exists.
     d. Hence, if $A \rightarrow B$, then A -->> B.
  2. **Intersection Rule:** If A -->> B and A -->> C, then A -->> B∩C.
  3. **Transitive Rule:** If A -->> B and B -->> C, then A -->> B-C.
- Functional dependencies are a subset of multi-valued dependencies.
  This means that any rules for multi-valued dependencies apply to functional dependencies but rules for functional dependencies may not apply to multi-valued dependencies.
- Here's the algorithm to decompose a relation into Fourth Normal Form:
  Input: Relation R + FDs for R + MVDs for R
  Output: Decomposition of R into 4NF relations with lossless joins.
  Steps:
  1. Compute the candidate keys for R.
  2. Repeat until all relations are in 4NF:
     a. Pick any R' with nontrivial A -->> B that violates 4NF.
     b. Decompose R' into R1(A, B) and R2(A, Rest)
     c. Compute FDs and MVDs for R1 and R2.
     d. Compute keys for R1 and R2.
- **E.g. 24.** Consider the relation Apply(SSN, collegeName, hobby) and the fact that SSN -->> collegeName (cName). Decompose Apply such that all relations are in 4NF.

  Solution:
  A1(SSN, cName)
  A2(SSN, hobby)

- **E.g. 25.** Consider the relation Apply(SSN, collegeName, date, major, hobby) and the fact that
  SSN, cName → date
  SSN, cName, date -->> major
  Decompose Apply such that all relations are in 4NF.

  Solution:
  A1(SSN, cName, date, major)
  A2(SSN, cName, date, hobby)

  We need to break up A1 and A2 further. I'll break A1 into A11 and A12 and A2 into A21.

  A11(SSN, cName, date)
  A12(SSN, cName, major)
  A21(SSN, cName, hobby)

Tutorial Notes:
E.g. 1. Given R(A, B, C, D) and the FDs
A → BC
AC → D
D → ABC
C → A

List out all the candidate keys.

Soln:
First, we try to look for any attribute(s) that are not on the RHS on any FD. If such attribute(s) exist, we know that all candidate keys must include those attribute(s) since the only way to derive those attribute(s) is from themselves. However, in this example, all attributes are listed on the RHS of some FD.

We see that A^+ = {A, B, C, D}, so {A} is a candidate key.
We see that C^+ = {A, B, C, D}, so {C} is a candidate key.
We see that D^+ = {A, B, C, D}, so {D} is a candidate key.

Hence, the candidate keys are {A, C, D}.
Lossy Decomposition:

- The decomposition of relation R into R1 and R2 is **lossy** when the join of R1 and R2 does not yield the same relation as in R.
- **Note:** A **lossy decomposition** does not necessarily mean that you lost data. You could also have gained false/incorrect data when you join the tables. It simply means that when you join the tables, it does not yield the same relation as the original table.
- One of the disadvantages of decomposition into two or more relations is that some information is lost during retrieval of original relation or table.
- Decomposition is **lossless** if it is feasible to reconstruct relation R from decomposed tables using Joins.

- **E.g. 2.** Consider the table below:

R

utorid	**name**	**grade**
g3tout	Amy	91
g4foobar	David	78
c0zhang	David	85

Suppose we broke R into the following 2 tables:

R1

utorid	**name**
g3tout	Amy
g4foobar	David
c0zhang	David

R2

name	**grade**
Amy	91
David	78
David	85

Lets see what happens when we join R1 and R2.

utorid	**name**	**grade**
g3tout	Amy	91
g4foobar	David	78
g4foobar	David	85
c0zhang	David	78
c0zhang	David	85

We see that we get 2 extra rows, the underlined rows, that weren't there before. This is an example of a lossy decomposition. Notice that we didn't lose any of the original data but we got false/incorrect data.

First Normal Form (1NF):

- For a table to be in 1NF, every cell in the relation can only take on a single value.
- **E.g. 3.** Consider the table below:

SID	**Major**
123	Math, Computer Science

Since there are 2 values under the Major column, this is not in 1NF. To fix it, we can do this:

SID	**Major**
123	Math
123	Computer Science

Second Normal Form (2NF):

- For a table to be in 2NF:
  1. It must already be in 1NF and
  2. It must not have any **partial dependencies**.
- A **partial dependency** occurs when non-prime attributes depend on a proper subset of the prime attributes.
  I.e. Let P be a proper subset of the prime attributes and let NP be some non-prime attributes. If we have P → NP, we have partial dependency.
- Here are the steps on how you can decompose R into smaller relations that are in 2NF:
  1. Identify all the candidate keys.
  2. Identify the prime and non-prime attributes.
  3. Identify the partial dependencies.
  4. Decompose the relation for the candidate keys and all partial dependencies.
- **E.g. 4.** Given R(A, B, C, D, E) and the FDs
  AB → C
  D → E

  Determine if R is in 2NF, and if it isn't decompose it so that the relations are in 2NF.

  Soln:
  Notice how the RHS of the FDs do not contain A, B and D. Hence, we know that all candidate keys must contain at least A, B and D. The closure of ABD is {A, B, C, D, E}, so in this case, the candidate key is {A, B, D}.

  Since the candidate key is {A, B, D}, the prime attributes are A, B and D and the non-prime attributes are C and E.

  We see that we have 2 partial dependencies:
  1. AB → C
  2. D → E

  I will decompose R into:
  R1(A, B, D) ← Relation with the prime attributes
  R2(A, B, C)
  R3(D, E)

Third Normal Form (3NF):

- A table is in 3NF if:
    1. It is in 2NF and
    2. It must not have any **transitive dependencies**.
- If you have a NP → NP, you have a transitive dependency.
- Here are the steps on how you can decompose R into smaller relations that are in 3NF:
    1. Identify all the candidate keys.
    2. Identify the prime and non-prime attributes.
    3. Identify the partial dependencies and the transitive dependencies.
    4. Decompose the relation for the candidate keys, all partial dependencies and all transitive dependencies.
- **E.g. 5.** Given R(A, B, C, D, E, F, G, H, I, J) and the FDs
  AB → C
  AD → GH
  BD → EF
  A → I
  H → J

  Determine if R is in 3NF, and if it isn't decompose it so that the relations are in 3NF.

  Soln:
  The candidate key is {A, B, D}.
  Hence, the prime attributes are A, B and D.
  The non-prime attributes are C, E, F, G, H, I, J.
  We see that
    - AB → C is a pd
    - AD → GH is a pd
    - BD → EF is a pd
    - A → I is a pd
    - H → J is a td.

  We can decompose R into:
  R1(A, B, C)
  R2(A, D, G, H)
  R3(B, D, E, F)
  R4(A, I)
  R5(H, J)
  R6(A, B, D)
- **Note:** If you have (P+NP) → NP, that is also a 3NF violation.
  E.g. If we have the FD AH → J, we know that A is a prime attribute, but H is a non-prime attribute and J is also a non-prime attribute. This would still be a transitive dependency and would violate 3NF.

Boyce-Codd Normal Form (BCNF/3.5NF):

- A table is in BCNF if:
    1. It is in 3NF and
    2. For every non-trivial FD X → Y, X must be a super key.
- **E.g. 6.** Suppose we have R(A, B, C, D, E, F) and the FDs
  A → B
  CD → E
  AC → F

Is R in BCNF?

Soln:
Let's look at A → B. The closure of A is {A, B}. Hence, A is not a super key, which means that R is not in BCNF.

- For every X → Y that violates BCNF, we create 2 tables
  R1(R - Y)
  R2(X, Y)

More Examples:

E.g. 7. Create an instance of R(A, B, C, D, E) that violates the FD ABC → DE.

Soln:

A	B	C	D	E
1	2	3	4	5
1	2	3	5	7

E.g. 8. Suppose we have a relation R(A, B, C, D, E). Does the instance below violate the FD DB → A?

A	B	C	D	E
5	3	2	1	6
5	8	3	1	2

Soln:
No, because A has the same values for all rows. It doesn't matter what DB is, it will always uniquely determine A.

E.g. 9. Suppose we have R(A, B, C, D, E) and the FDs
A → BD
D → E

Is R in BCNF?

Soln:
No. Let's look at the closure of D. It is {D, E}. Hence, D is not a super key, which means that R is not in BCNF.

E.g. 10. Given R(A, B, C, D) and the FDs
A → B
AC → D
Is R in BCNF? If it's not, decompose it so that the relations are in BCNF.

Soln:
We see that R is not in BCNF because in A → B, A is not a super key.
We will decompose R into
R1(A, C, D)
R2(A, B)

Summary of Normal Forms:

Form	Explanation	Decomposition
1NF (First Normal Form)	Each cell can only contain 1 value.	If a cell contains multiple values, create a new row for each value.
2NF (Second Normal Form)	Must be in 1NF. Cannot have partial dependencies (pds).	Identify all the candidate keys. Identify the prime and non-prime attributes. Identify the partial dependencies. Decompose the relation for the candidate keys and all pds.
3NF (Third Normal Form)	Must be in 2NF. Cannot have transitive dependencies (tds).	Identify all the candidate keys. Identify the prime and non-prime attributes. Identify the partial dependencies and the transitive dependencies. Decompose the relation for the candidate keys, all pds and all tds.
BCNF (Boyce-Codd Normal Form)	Must be in 3NF. For each non-trivial FD $X \rightarrow Y$, X must be a super key.	decompose (R, $X \rightarrow Y$): R1 (R - Y) R2 (X + Y) project FDs onto R1 and R2 recursively call decompose on R1 and R2 for BCNF violations OR decompose (R, $X \rightarrow Y$): R1 (X+) R2 (R - (X+ - X)) project FDs onto R1 and R2 recursively call decompose on R1 and R2 for BCNF violations
4NF (Fourth Normal Form)	Must be in BCNF. Cannot have multi-valued dependencies (mvds).	decompose (R, X -->> Y): R1 = XY R2 = X union (R - Y) Repeat on R1 and R2 until all relations are in 4NF