- Let $K \subseteq R$
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor*
- Superkey K is a **candidate key** if K is minimal

  Example: {*ID*} is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation
  - **Referenced** relation
  - Example: *dept_name* in i*nstructor* is a foreign key from *instructor* referencing *department*

- **Total participation** (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set

participation of *student* in *advisor r*elation is total

  - every *student* must have an associated instructor
- **Partial participation**: some entities may not participate in any relationship in the relationship set
  - Example: participation of *instructor* in *advisor* is partial

Choice of Primary key for Binary Relationship

Many-to-Many relationships. The preceding union of the primary keys is a minimal superkey and is chosen as the primary key

One-to-Many relationships . The primary key of the "Many" side is a minimal superkey and is used as the primary key

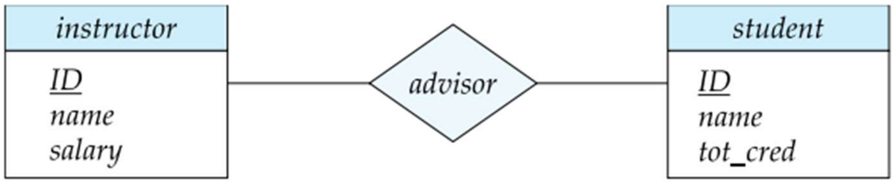
Many-to-one relationships. The primary key of the "Many" side is a minimal superkey and is used as the primary key

One-to-one relationships. The primary key of either one of the participating entity sets forms a minimal superkey, and either one can be chosen as the primary key

- In E-R diagrams, a weak entity set is depicted via a double rectangle
- We underline the discriminator of a weak entity set with a dashed line
- The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a double diamond
- In general, a weak entity must have total participation in its identifying relationship set, and the relationship is many-to-one towards the identifying entity set
- Primary key for *section* – (*course_id, sec_id, semester, year*)

- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
- Example: schema for relationship set *advisor*

advisor = (<u>*s_id, i_id*</u>)



- Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the "many" side, containing the primary key of the "one" side
  - Example: Instead of creating a schema for relationship set *inst_dept*, add an attribute *dept_name* to the schema arising from entity set *instructor*
- For one-to-one relationship sets, either side can be chosen to act as the "many" side
  - That is, an extra attribute can be added to either of the tables corresponding to the two entity sets

First Normal Form

- Domain is **atomic** if its elements are considered to be indivisible units
  - Example of non-atomic domains:

    Variable number of banking transactions of an account (sometimes called a repeating group)
  - Variable number of locations of a department

- A relational schema *R* is in **first normal form** if the domains of all attributes of *R* are atomic

Keys and Functional Dependencies

- K is a **superkey** for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
  - One of the candidate keys is designated to be the **primary key,** and the others can be called **secondary keys**
- In general, any data field other than the primary key can be called a **secondary key** (particularly for search and indexing purposes)
  - A secondary key may or may not uniquely identify a tuple (i.e., may or may not be a superkey)

Keys and Functional Dependencies

- A **prime attribute** is a member of *some* candidate key
  - If there is only one candidate key – the primary key – then a **prime attribute** is an attribute that is member of the primary key
- A **nonprime attribute** is not a prime attribute — that is, it is not a member of any candidate key
  - If there is only one candidate key – the primary key – then a **nonprime attribute** is an attribute that is not a member of the primary key

Lossless Decomposition

- We can use functional dependencies to show when certain decomposition are lossless
- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless decomposition if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition

Second Normal Form

- A relation schema R is in **second normal form (2NF)** if it is in first normal form, and every nonprime attribute A in R is fully functionally dependent on the primary key (assuming it is the only candidate key)

Third Normal Form

- A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no nonprime attribute A in R is transitively dependent on the primary key (assuming it is the only candidate key)

General Normal Form Definitions for Multiple Candidate Keys

- A relation schema R is in **second normal form (2NF)** if it is in 1NF, and every nonprime attribute A in R is fully functionally dependent on *every* candidate key of R

General Definition of Third Normal Form

- Definition:
  - **Superkey** of relation schema R - a set of attributes S of R that contains a candidate key of R
  - A relation schema R is in **third normal form (3NF)** if whenever a nontrivial functional dependency $X \rightarrow A$ holds in R, then either:
    - (a) X is a superkey of R, or
    - (b) A is a prime attribute of R (more precisely, A-X is a prime attribute of R)

Alternative Definition of Third Normal Form

- **We can restate the definition as:**

  A relation schema R is in **third normal form (3NF)** if every nonprime attribute in R meets both of these conditions:

  - It is fully functionally dependent on every candidate key of R
  - It is non-transitively dependent on every candidate key of R

  Note that stated this way, a relation in 3NF also meets the requirements for 2NF

Boyce-Codd Normal Form

- A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form

  $\alpha \rightarrow \beta$

  where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R
- If we call the left hand side of an FD a determinant, BCNF roughly says that every determinant is a superkey

Decomposing a Schema into BCNF

- Let R be a schema R that is not in BCNF. Let $\alpha \rightarrow \beta$ be the FD that causes a violation of BCNF, i.e. α is not a superkey of R.
- We decompose R into:
  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$

Third Normal Form Revisited

- A relation schema R is in **third normal form (3NF)** if for all:

  $$\alpha \rightarrow \beta \text{ in } F^+$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

  (**NOTE***:* the third condition does not say that a single candidate key must contain all the attributes in $\beta - \alpha$; each attribute A in $\beta - \alpha$ may be contained in a *different* candidate key)

Closure of a Set of Functional Dependencies

- We have seen that the set of **all** functional dependencies logically implied by *F* is the **closure** of *F,* and *we* denote the *closure* of *F* by F^+
- We can compute F^+ by repeatedly applying **Armstrong's Axioms:**
  - **Reflexive rule:** if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
  - **Augmentation rule:** if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
  - **Transitivity rule:** if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- These rules are
  - **Sound** -- generate only functional dependencies that actually hold, and
  - **Complete** -- generate all functional dependencies that hold

Closure of Functional Dependencies

- Additional rules:
  - **Union rule**: If $\alpha \rightarrow \beta$ holds *a*nd $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds
  - **Decomposition rule**: If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds
  - **Pseudotransitivity rule**: If $\alpha \rightarrow \beta$ holds *a*nd $\gamma\beta \rightarrow \delta$ holds, then $\gamma\alpha \rightarrow \delta$ holds
- The above rules can be inferred from Armstrong's axioms

Procedure for Computing F^+

- The following computes the closure of a set of functional dependencies *F*

```
F + = F
apply the reflexivity rule  /* Generate all trivial dependencies */
repeat
    for each functional dependency f in F+
        apply the augmentation rule on f
        add the resulting functional dependencies to F +
    for each pair of functional dependencies f1 and f2 in F +
        if f1 and f2 can be combined using transitivity
            then add the resulting functional dependency to F +
until F + does not change any further
```

Closure of Attribute Sets

- Given a set of attributes α, define the ***closure*** of α **under** *F* (denoted by α^+) as the set of attributes that are functionally determined by α under *F*
- Algorithm to compute α^+, the closure of α under *F*

```
result := α;
repeat
    for each functional dependency β → γ in F do
        begin
            if β ⊆ result then result := result ∪ γ ;
        end
until (result does not change)
```

Extraneous Attributes

- An attribute of a functional dependency in F is **extraneous** if we can remove it without changing F^+
- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F
  - **Remove from the left side**: Attribute A is **extraneous** in α if
    - $A \in \alpha$ and
    - $F \Rightarrow (F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\} = F'$,

      replacing the functional dependency $\alpha \rightarrow \beta$ by a new functional dependency by taking out A from the left-hand side
    - i.e., it is assumed possible to replace a weaker FD by a stronger FD
  - **Remove from the right side**: Attribute A is **extraneous** in β if
    - $A \in \beta$ and
    - The set of functional dependencies

      $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\} \Rightarrow F$
    - i.e., it is assumed possible to replace a stronger FD by a weaker FD
- *Note:* implication in the opposite direction is trivial in each of the cases above, since a "stronger" FD always implies a "weaker" one

Testing if an Attribute is Extraneous

- Let R be a relation schema and let F be a set of functional dependencies that hold on R . Consider an attribute in the functional dependency $\alpha \rightarrow \beta$.
- To test if attribute $A \in \beta$ is extraneous in β
  - Consider the set (i.e., removing A from the FD)

    $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,
  - check that α^+ contains A under F'; if it does, A is extraneous in β
- To test if attribute $A \in \alpha$ is extraneous in α
  - Let $\gamma = \alpha - \{A\}$. Check if $\gamma \rightarrow \beta$ can be inferred from F.
    - Compute γ^+ using the dependencies in F
    - If γ^+ includes all attributes in β, then A is extraneous in α

Canonical Cover

- To compute a canonical cover for F:

  $F_c = F$

  repeat

  Use the union rule to replace any dependencies in F of the form

  $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$

  Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β

  /* Note: test for extraneous attributes done using $F_{c,}$ not F */

  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$ in F_c.

  until (F_c does not change)

- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

BCNF Decomposition Algorithm

result := {R};
done := false;
while (not *done)* **do**
if (there is a schema R_i in *result* that is not in BCNF)
then begin
let $\alpha \rightarrow \beta$ be a nontrivial functional dependency
that holds on R_i such that α is not a superkey of R_i
and $\alpha \cap \beta = \emptyset$;
result := (*result* – R_i) $\cup$ ($R_i - \beta$) $\cup$ (α, β);
end
else *done* := **true;**

<u>Note:</u>

- If $\alpha \cap \beta \neq \emptyset$, e.g., $\alpha \cap \beta = \gamma$, then ($R_i - \beta$) would exclude γ, losing the information on γ in $R_i - \beta$, and α is incomplete in $R_i - \beta$ which would be undesirable (see next example)

3NF Decomposition Algorithm

```
Let F_c be a canonical cover for F;
i := 0;
for each functional dependency α → β in F_c do
  if none of the schemas R_j, 1 ≤ j ≤ i contains α β
        then begin
              i := i + 1;
              R_i := α β
           end
if none of the schemas R_j, 1 ≤ j ≤ i contains a candidate key for R
  then begin
           i := i + 1;
           R_i := any candidate key for R;
        end
repeat /* Optionally, remove redundant relations */
   if any schema R_j is contained in another schema R_k
      then /* Delete R_j */
        R_j := R_i;
        i := i-1;
until no more R_j's can be deleted
return (R_1, R_2, ..., R_i)
```