- **Problems:**
  - **Insertion Anomaly:** we can't store info about a new movie if the screening cinema and time are unknown **e.g. Infinitiy War have not come out yet, dk info**
  - **Deletion Anomaly:** if we delete all NC16-rated movie, we lose information about the cinema Vivocity
  - **Update Anomaly:** if the rating of a movie changes, we have to be careful of inconsistent updates (*update on all places*)

e.g. change Capt Marvel in 1st row to NC16, but now Capt Marvel in last row is still PG13

Functional Dependency

Functional Dependencies (FD) are constraint on schemas that specify that the values for a certain set of attributes determine unique values for another set of attributes

Let r be a relation instance of relation schema R

- **Satisfies**
  - r satisfies FD $a \to b$ *if* for every pair of tuples t_1 and t_2 in r such that $\pi_a(t_1) = \pi_a(t_2)$, it is also true that $\pi_b(t_1) = \pi_b(t_2)$
- **Holds**
  - An FD f holds on R *if and only if* for any relation instance r of R, r satisfies R
- **Violates**
  - r violates an FD f *if* r does not satisfy f
- **Legal Instance**
  - r is a legal instance of R *if* r satisfies all FDs that holds on R

- Trivial FD
  - An FD $a \to b$ is a trivial FD *if* ~~a~~ b is a subset of ~~b~~ a (i.e., $a \supseteq b$)

e.g. m -> m, t -> m; trivial is always true; superset notation

- Non-Trivial FD
  - An FD $a \to b$ is a non-trivial FD *if* it is NOT a trivial FD (i.e., $a \not\supseteq b$)

e.g. movie -> movie rating; e.g AC -> AF

- Completely Non-Trivial FD
  - An FD $a \to b$ is a completely non-trivial FD *if* a and b have completely different attributes (i.e., $a \cap b = \emptyset$)

e.g. movie -> rating; Completely non-trivial is also non-trivial (above bullet)

- Let F and G denote sets of FDs, and f denote an FD
- **Logical Implication:**
  - F logically implies (or implies, denoted $F \vDash f$) f *if* every relation instance r of R that satisfies the FDs F also satisfies the FD f
  - **More generally:** $F \vDash G$ *if* $F \vDash g$ for each $g \in G$
- **FD Closure:**
  - The closure of F is the set of all FDs implied by F
  - $F^+ = \{f \mid F \vDash f\}$
- **Equivalence:**
  - F is equivalent to G (denoted $F \equiv G$) *if* $F^+ = G^+$
  - **In other words:** $F \vDash G$ and $G \vDash F$

Armstrong Axiom

- Let $a, b, c \subseteq R$
  - **Reflexivity:** if $b \subseteq a$ then $a \to b$ — trivial
  - **Augmentation:** if $a \to b$ then $ac \to bc$
  - **Transitivity:** if $a \to b$ and $b \to c$ then $a \to c$
- **Properties:**
  - **Sound:** any derived FD is implied by F
  - **Complete:** all FDs in F^+ can be derived
  - **Extension:**
    - **Union:** if $a \to b$ and $a \to c$ then $a \to bc$
    - **Decomposition:** if $a \to b$ then $a \to b'$ where $b' \subseteq b$
    - Specific Case: if $a \to bc$ then $a \to b$ and $a \to c$

e.g. a -> bc, Therefore, a -> b or a -> c; if A -> C, means can add anything on left, then AB -> C, becos AB -> BC then decompose become AB -> C

Key, Prime attribute

- **Superkey**
  - A set of attributes a is a superkey of schema R (with FDs F) *if* $F \vDash a \to R$

i.e. based on all sets of FD in F, i realised that i can determine all attributes in R; means myself can determines all attributes of relation; e.g. R(ABCD)

- **Key**
  - A set of attributes a is a key of schema R *if*
    - a is a superkey, *and*
    - No proper subset of a is a superkey (i.e., for each $b \subset a$, $F \nvDash b \to R$)

i.e. candidate key; if AB -> ABCD, then AB is a superkey; Based on AB alone, it can uniquely identifies all ABCD

- **Prime Attribute**
  - An attribute $A \in R$ is a prime attribute *if* A is contained in some key of R
  - Otherwise it is a nonprime attribute

Find all candidate keys, then from there take all attributes from all candidate keys; To find candidate keys, we find closure of attribute to see if it contains all attributes in R; e.g. AB is a key, AC is a key, therefore prime attributes are A, B, and C

Attribute Closure

- Let $a \subseteq R$ and F be a set of FDs that hold on R
- The closure of a (with respect to F), denoted by a^+, is the set of attributes that are functionally determined by a with respect to F
  - $a^+ = \{A \in R \mid F \vDash a \to A\}$
- Note that $F \vDash a \to b$ *if and only if* $b \subseteq a^+$ (w.r.t. F)

Redundant FD

Compute attribute closure of A becos now we want remove A->B, we need make sure whats implied by A->B still remains. Hence we compare A+ before remove A->B, and A+ after remove A->B. If both same means A->B redundant

- **Check:** how to check if $a \to B$ is redundant in F
  - Compute a^+ w.r.t. $F - \{a \to B\}$
  - $a \to B$ is redundant in F *if* $B \in a^+$
- **Example:** $F = \{A \to B,\ A \to C,\ B \to A,\ B \to C,\ C \to A\}$
  - $A \to B$ is non-redundant: $A^+ = AC$ w.r.t. $F - \{A \to B\}$
  - $A \to C$ is redundant: $A^+ = ABC$ w.r.t. $F - \{A \to C\}$

Remove away a->B still can get same closure, then a->B is redundant

Redundant Attribute

AB+ = ABC
B+ also = ABC
therefore A redundant

AB+ = ABC
A+ also = ABC
therefore B redundant

Minimal Cover

- A minimal cover for a set F of FDs is a set G of FDs with the following properties:
  - Every FD in G is of the form $a \to A$ — Singleton
  - For each FD $a \to A$ in G, a has NO redundant attributes
  - There are NO redundant FDs in G
  - G and F are equivalent

e.g. if ab -> c, a-> c, then b is redundant

- **Input:** A set of FDs F
- **Output:** A minimal cover for F

Steps: 1) Decompose RHS to singleton 2) Remove away all redundant attributes in all FD 3) Remove any redundant FD

```
1.  initialize  G = ∅
2.  for each (FD  a → B1 ... Bn ∈ F)
3.    G = G ∪ {a → Bi | i ∈ [1, n]}
4.  for each (FD  a → B ∈ G)
5.    initialize  a' = a
6.    for each ( A ∈ a ) do
7.      if (B ∈ (a' − A)+  w.r.t.  G) then
8.        replace a' → B  in  G  by  a' − A → B
9.        a' = a' − A
10. for each (FD  a → B ∈ G)
11.   if (B ∈ a+  w.r.t.  G − {a → B}) then
12.     remove  a → B  from  G
13. return  G
```

Decompose RHS to singleton; a.g. AB -> BCD become AB-> B, AB-> C, AB-> D

e.g. G = {AB -> B, AB -> C, AB -> D}; Remove redundant attributes; e.g. for AB -> B check if can remove A or B ornot; check whether F |= B -> C (A redundant) or F |= A ->C (B redundant)

Remove redundant FD; e.g. G = {AB -> B, AB -> C, AB -> D} check can remove any of the FD ornot

```
with StudentInfo as (
    select stuid, count(*) as numQ, max(week) as lastWk
    from Presenters
    group by stuid
    union
    select stuid, 0, 0
    from Students
    where stuid not in (select stuid from Presenters)
),
StudentSet as (
    select stuid
    from StudentInfo
    order by numQ, lastWk, stuid
    limit 2
)
select L1.cname from likes L1 group by L1.cname
having count(L1.cname) = (
select max(cc.cnt) from (select count(L2.cname) cnt
from likes L2 group by L2.cname) cc
)
```

Schema Decomposition

How to split a table so all attributes are still captured

- The decomposition of schema R is a set of schemas $\{R_1, R_2, \ldots, R_n\}$ such that each $R_i \subseteq R$ and $R = R_1 \cup R_2 \cup \cdots \cup R_n$

decomposition is a subset

- If r is a relation of R, then r is decomposed into $\{r_1, r_2, \ldots, r_n\}$ where each $r_i = \pi_{R_i}(r)$
- **Properties:**
  - Decomposition must preserve information

Use natural join on decomposed table should

    - Data in original relation ≡ Data in decomposed relations
    - Crucial for correctness!
  - Decomposition should preserve FDs
    - FDs in original schema ≡ FDs in decomposed schemas
    - Facilitates checking of FD violations without performing joins

- If $\alpha \to \beta$ is a completely non-trivial FD that holds on R, then the decomposition of R into $\{R - \beta, \alpha\beta\}$ is a lossless-join decomposition

i.e. α∩β = φ

Projection

- The projection of F on a (denoted by F_a) is the set of FDs in F^+ that involves only attributes in a
  - $F_a = \{b \to c \in F^+ \mid bc \subseteq a\}$
- **Example:**
  - Consider $R(A, B, C)$ with FDs $F = \{A \to B, B \to C, C \to B\}$

Another way to look at: In F+, remove all FD that you are not interested in, i.e. those with attributes other than AB if you finding F(AB)

Algorithm 3: Computing FD Projections

- **Input:** A set of attributes $a \subseteq R$ and a set of FDs F on R
- **Output:** F_a e.g. F(AB)

```
1. initialize res = ∅
2. for each (b ⊆ a  such that  b ≠ ∅)  means for A, B, AB
3.     b+  (w.r.t.  F)  // Algorithm 1  find closure of A+, B+, AB+ e.g A+ = ABC
4.   res = res ∪ {b → (b+ ∩ a)}  find closure ∩ AB e.g. if A -> ABC, take ABC ∩ AB which is A -> AB then union all resultants
5. return res
```

Dependency Preserving

- Decomposition $\{R_1, R_2, \ldots, R_n\}$ of R is dependency-preserving *if* $(F_{R_1} \cup F_{R_2} \cup \cdots \cup F_{R_n})$ is equivalent to F
  - $(F_{R_1} \cup F_{R_2} \cup \cdots \cup F_{R_n}) \equiv F$
  - $(F_{R_1} \cup F_{R_2} \cup \cdots \cup F_{R_n})^+ \equiv F^+$
  - $F_{R_1} \cup F_{R_2} \cup \cdots \cup F_{R_n} \vDash F$ and $F \vDash \cup_{i=1}^{n} F_{R_i}$

Algorithm 4: Dependency-Preserving Decomposition

- **Input:** A decomposition $\{R_1, \ldots, R_n\}$ of R with FDs F e.g. R(ABC) F= {B -> C, AC -> B}
- **Output:** *YES (if dependency-preserving) / NO (otherwise)*

```
1. for each (Ri ∈ {R1, ..., Rn}) R1(AB), R2(BC)
     compute  F_Ri  // Algorithm 3   Projection  F(AB) = {A -> A, B -> B, AB -> AB}  F(BC) = {B -> BC, C -> C, BC -> BC}
3. let  G = F_R1 ∪ ... ∪ F_Rn   G = {A -> A, B -> B, AB -> AB, B -> BC, C -> C, BC -> BC}
4. for each (FD a → b ∈ F)
5.     compute  a+  w.r.t.  G  // Algorithm 1   For B -> C: B+ = BC Since C in B+, then ok   For AC -> B: AC+ = AC Since B not in AC+, return NO
6.     if  (b ⊈ a+)  then  return NO
7. return YES
```

Basically: 1. Identify R, R1, R2, and F for R 2. Find projection of R1, R2 3. Combine projections together 4. Check and see original F preserves in combined projections ornot

1NF: Domain of each attribute contains only atomic values
2NF: Satisfy 1NF
Every non-prime attribute of the relation is dependent on the whole of every candidate key
NOTE: does not put any restriction on the non-prime to non-prime attribute dependency

BCNF

- Every project is sponsored by at least one department
- Each sponsorship has a "since" attribute & might be monitored by 0 or more employees
- Each monitoring has a "until" attribute


```
create table Projects (
    pid         char(20),
    startedOn   date,
    primary key (pid)
);

create table Departments (
    did     char(20),
    dname   char(30),
    primary key (did)
);

create table Sponsors (
    pid         char(20)
    references Projects,
    did         char(20)
    references Departments,
    since       date,
    primary key (pid,did)
);

create table Employees (
    eid     char(20),
    ename   char(30),
    primary key (eid)
);

create table Monitors (
    eid     char(20) references Employees,
    pid     char(30),
    did     char(30),
    until   date,
    primary key (eid,pid,did),
    foreign key (pid,did) references Sponsors (pid,did)
);
```

By: Gan Chin Yao | 2019 May

BCNF

Definition: No data redundancy in BCNF

- Consider a relation schema R with FDs F
- R is in Boyce-Codd Normal Form (BCNF) *if* for every non-trivial FD $a \to A$ in F — therefore A is definitely non-prime attributes; A is singleton; non-prime attributes are dependent on superkey
  - a is a superkey of R
- **Violation:**
  - A non-trivial FD $a \to b$ that holds on R is said to violate BCNF *if*
    - a is not a superkey of R
- **Decomposition:**
  - A decomposition $\{R_1, R_2, \dots, R_n\}$ of R (with FDs F) is in BCNF *if* each R_i is in BCNF (w.r.t. F_{R_i}) — No need reference to original F
- **Algorithm 5:** BCNF Test for R_i — look for FD that violate BCNF
  - **Input:** F is a set of FDs that hold on a schema R and R_i is either R or a decomposed schema of R
  - **Output:** A completely non-trivial FD that violates BCNF if R_i is not in BCNF; otherwise, *null*

```
1. if (Ri has exactly 2 attributes) then return null
2. for each (a ⊆ R such that a ≠ ∅)
3.   let X = a+ ∩ Ri (w.r.t. F)
     // a → X is an FD that holds on Ri   (non-trivial FD)
4.   if (a ⊂ X ⊂ Ri) then return a → X − a   (non-trivial, not a superkey)
5. return null
```

- **Algorithm 6:** BCNF Decomposition
  - **Input:** Schema R with FDs F
  - **Output:** A BCNF decomposition of R

```
1. initialize δ = ∅;  i = 1;  temp = {R}
2. while (temp ≠ ∅)
3.   remove some R' from temp
4.   f = BCNFTest(F, R') // Algorithm 5   look for FD that violate BCNF
5.   if (f = null) then
6.     δ = δ ∪ {R'}
7.   else
8.     let f be a → b
       let c = R' − b
       temp = temp ∪ {Ri(ab), Ri+1(c)}   // for next check
11.    i = i+2
12. return δ
```

There is a violation of BCNF. Here make sure it doesnt violate BCNF.

guaranteed to have lossless join decomposition becos its decomposed based on FD

- **Properties:**
  - BCNF decompositions are lossless-join decompositions — **Not guaranteed losslessjoin unless algo6 used**
  - However, not all schema has a dependency-preserving BCNF decompositions — **No data redundancy**
  - BCNF decompositions are not necessarily dependency-preserving

3NF

- Consider a relation schema R with FDs F
- R is in Third Normal Form (3NF) *if* for every non-trivial FD $a \to A$ in F
  - a is a superkey of R **OR** // *BCNF*
  - A is a prime attribute // *but less restrictive*
- **Violation:** A is RHS, not LHS. BCNF guarantees no data redundancy, but not all schema has BCNF. Therefore 3NF is more relax. If a is not superkey, we allow A to be prime attribute
  - A non-trivial FD $a \to b$ that holds on R is said to violate BCNF *if*
    - a is not a superkey of R ***AND***
    - A is a non-prime attribute
- **Decomposition:**
  - A decomposition $\{R_1, R_2, \dots, R_n\}$ of R (with FDs F) is in 3NF *if* each R_i is in 3NF (w.r.t. F_{R_i})

3NF property: All schema can be decompose to 3NF, 3NF is not cfm lossless join , 3NF not cfm dependency preserving, some data may be redundant (as compared to BCNF definitely no redundant data)

- **Algorithm 7:** 3NF Decomposition
  - **Input:** Schema R with FDs F which is a *minimal cover*
  - **Output:** A dependency-preserving and lossless-join 3NF decomposition of R

```
1. initialize δ = ∅
2. apply union rule to combine FDs in F with same L.H.S into a single FD
3. let G = {f1, f2, ..., fn} be the resultant set of FDs
4. for each (FD fi of the form ai → bi in G)
5.   create a relation schema Ri(aibi) for FD fi
6.   insert the relation schema Ri(aibi) into δ
7. choose a key K of R and insert a relation schema Rn+1(K) into δ
8. remove redundant relation schema from δ as follows:
9.   delete Ri from δ if ∃Rj ∈ δ · i ≠ j ∧ Ri ⊆ Rj
10. return δ
```

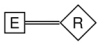
- **Algorithm 7:** Properties
  - Each $R_i \in \delta$ is in 3NF
  - δ is a dependency-preserving decomposition
  - δ is a lossless-join decomposition
  - The decomposition δ produced by Algorithm 7 (*i.e., synthesis approach*) may not be unique
    - Depends on choice of minimal cover
    - Depends on choice of redundant relation schema being removed

3NF is not unique

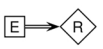
ER

0 or more: Partial participation constraint

Each instance of E participates in at most one instance of R **Key constraint**



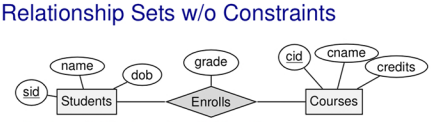
Each instance of E participates in at least one instance of R **Total participation constraint**



Each instance of E participates in exactly one instance of R

E is a weak entity set with identifying owner E' & identifying relationship set R

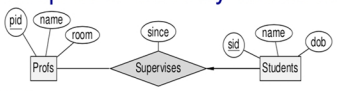
Relationship Sets w/o Constraints



Many-to-many: just take all primary key and stick them tgt

```
create table Enrolls (
  sid    char(20) references Students,
  cid    char(5),
  grade  numeric,
  primary key (sid, cid),
  foreign key (cid) references Courses);
```

Relationship Sets with Key Constraints



Better approach

- First approach: Represent `Supervises` using a separate table
  - **Profs** (pid, name, room)
  - **Students** (sid, name, dob)
  - **Supervises** (sid, pid, since)
- Second approach: Combine `Supervises` & `Students` into one table — Works becos we can just assign null to student if student dont participate
  - **Profs** (pid, name, room)
  - **SupervisedStudents** (sid, name, dob, pid, since)

Relationship Sets with Key Constraints (cont.)

Cannot make relationship attribute as key
The attribute must be present in some entity

First approach:

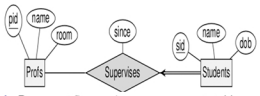
```
create table Supervises (
  sid    char(20),
  pid    char(7),
  since  date,
  primary key (sid),
  foreign key (sid)
    references Students,
  foreign key (pid)
    references Profs);
```

primary key cannot take both pid and sid cos otherwise will break "at most one" property of Students

Second approach:

```
create table SupervisedStudents (
  sid    char(20),
  name   char(30),
  dob    date,
  pid    char(7),
  since  date,
  primary key (sid),
  foreign key (pid)
    references Profs);
```

Key & Total Participation Constraints



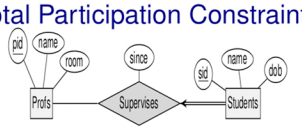
First approach: Represent `Supervises` as a separate table.

```
create table Supervises (
  sid    char(20) primary key,
  pid    char(7),
  since  date,
  foreign key (sid) references Students,
  foreign key (pid) references Profs);
```

Total participation constraint of `Students` w.r.t. `Supervises` is not captured by database schema!

Means i can insert a new record to Student, but dont insert a new record to Supervises. The previous one ok becos it is at most one, means can dont insert. But this is exactly one, must insert

Key & Total Participation Constraints



Better approach

Second approach: Combine `Students` & `Supervises` into a single table

```
create table SupervisedStudents (
  sid    char(20),
  name   char(30),
  dob    date,
  pid    char(7) not null,
  since  date,
  primary key (sid),
  foreign key (pid) references Profs);
```

Must put not null

Weak Entity Sets

- Weak entity set & its identifying relationship set are represented by a single relation

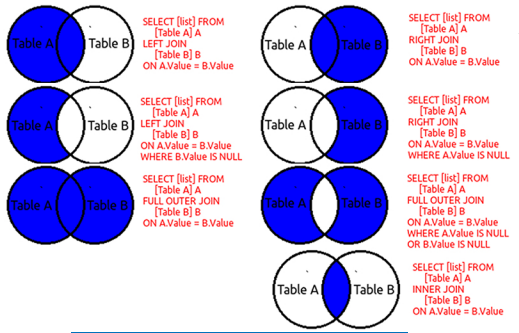
Dont need put not null becos primary key unique and not null

```
create table Books (
  isbn    char(30),
  title   char(50),
  author  char(60),
  primary key (isbn) );
```

```
create table BookChapters (
  number  integer,
  title   char(50),
  isbn    char(30),
  primary key (number, isbn),
  foreign key (isbn) references Books
  on delete cascade);
```

IS-A

- **Overlap constraint**: Can an entity belong to multiple subclasses?
  - A ISA hierarchy satisfies the overlap constraint if an entity in a superclass could belong to multiple subclasses
- **Covering constraint** Does an entity in a superclass have to belong to some subclass?
  - A ISA hierarchy satisfies the covering constraint if every entity in a superclass has to belong to some subclass




```
select      distinct select-list
from        from-list
where       where-condition
group by    groupby-list
having      having-condition
order by    orderby-list
limit       limit-specification
offset      offset-specification
```

Table is subset of another

```
where not exists (
    select 1 from
        (select Works.eid from Works
        where Works.pid = Projects.pid) as T1
        left outer join
        (select Specializes.eid from Specializes
        where Specializes.aid = 'A') as T2
        on T1.eid = T2.eid
    where T2.eid is null
)
```

exist / not exists

Returns *true* if the result subquery is non-empty; otherwise, *false*

```
where  exists (
       select  1
       from    Sells S
       where   S.rname = 'Corleone Corner'
       and     S.pizza = L.pizza
       );
```

if you dont use 'exists':

in

Subquery must return exactly one column

Returns *false* if result of subquery is empty; otherwise return the result of the boolean expression

$$((v = v_1) \text{ or } (v = v_2) \text{ or } \cdots \text{ or } (v = v_n))$$

where

- v denote the result of expression
- $\{v_1, v_2, \cdots, v_n\}$ denote the result of subquery

```
where  pizza in (
       select  pizza
       from    Sells
       where   rname = 'Corleone Corner'
       );
```

this subquery must return only 1 column otherwise its not meaningful to compare

any

Subquery must return exactly one column

Returns *false* if result of subquery is empty; otherwise return the result of the boolean expression

The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

$$((v \; op \; v_1) \text{ or } (v \; op \; v_2) \text{ or } \cdots \text{ or } (v \; op \; v_n))$$

```
select  distinct rname
from    Sells
where   rname <> 'Corleone Corner'
and     price > any ( /all
        select  price
        from    Sells
        where   rname = 'Corleone Corner'
        );
```

case

```
select name, case
        when marks >= 70 then 'A'
        when marks >= 60 then 'B'
        when marks >= 50 then 'C'
        else 'D'
end as grade
from   Scores;
select name, case
        when (first = 'pass') or (second = 'pass')
             or (third = 'pass') then 'pass'
        else 'fail'
end as result
from   Tests;
```

like

if '_ _ _ e' means exactly 4 chars ending with e
Can stick the % anywhere, it means 0 or more characters

```
select name from Customers where cname like '_ _ _%e';
```

coalesce

```
select name, coalesce(third,second,first) as result
from   Tests;
```

coalesce at least 2 argument. If all null, value is null
Can also use: coalesce(argument1, 'explicit value')

- **coalesce** returns the first non-null value in its arguments
- Null is returned if all arguments are null

nullif

```
select name, nullif(result,'absent') as stat
from   Tests;
```

- **nullif** ($value_1$, $value_2$)
- Returns *null* if $value_1$ is equal to $value_2$; otherwise returns $value_1$

```
ALTER TABLE table_name ADD COLUMN new_column_name TYPE;
ALTER TABLE table_name DROP COLUMN column_name;
ALTER TABLE table_name RENAME COLUMN column_name TO new_column_name;
ALTER TABLE table_name ALTER COLUMN column_name [SET DEFAULT value | DROP DEFAULT]
ALTER TABLE table_name ALTER COLUMN column_name [SET NOT NULL | DROP NOT NULL]
ALTER TABLE table_name ADD CHECK expression;
ALTER TABLE table_name RENAME TO new_table_name;
Operator precedence: (), op, ~, ^, v
Can insert many null into unique column
subquery in FROM must have an alias, e.g. select * from (select * from Sells) as S //ok
```

By: Gan Chin Yao | 2019 May