- Schema + Query Language Determine Application Program

- Goal: Have a schema let in the query simply avoids Redundancy, update anomalies, loss of info

Example:

Schema 1:  
Emp (eno, ename, bvr, sl1, cl1)  
Dept (dno, dname, floor, mgr)  
e1 or m1

Q1: Find all employees we make more than their manager:

Select E.ename  
From Emp E, Dept D, Emp E2  
where (D.mgr = E2.eno) and (E.dno = D.dno)  
and (E.sal > E2.sal)

Q2: Find all employees with a mes salary > 2x query salaries

Select (d.dname)  
From Emp E, Dept D  
where (D.dname = E.dno)  
Group by d.dname  
having D.max (E.sal) > 2*AVG(E.sal)

Schema 2:  
E0 (eno, ename, bvr, sl1, dno, dname, floor, mgr)

Q1: Select e.ename  
From E0 e1, E0 e2  
where (E1.bvr = E2.bvr) and (E1.sl1 > E2.sl1)

Q2: Select E.dname  
From E0 e  
Group by E.dno, E.dname  
having MAX(E.sal) > 2*AVG(E.sal)
* We get simpler queries, but we also get into trouble:

- **Redundancy**
  - Each dept is repeated once for each employee.
  - There is potential inconsistency (update anomalies).
  - We may change the location of a dept in one
    multiple leaving it with the old value in another.
  - A simple change is translated into multiple replacements.
    (e.g., change the manager of the "shoe" department.)

- **No independent existence:**
  - A dept cannot exist without employees.
  - When we delete the last employee of a dept, we
    automatically lose track of the dept.

**Objective of DB Design**

- No redundancy for space efficiency.
- Update integrity.
- Semantic clarity.
- Linguistic efficiency (the simpler the queries the better for
  both the user and query optimizer).
- Performance (binary relations imply most queries will have
  a large number of joins.)
Tools for DO design: Functional Dependencies & Normalization

**Functional Dependency**: Let $A, B$ be sets of attributes of $R$.

The $F_D$ $A \rightarrow B$ holds if a value for $A$ uniquely determines a value for $B$.

Formally: $A \rightarrow B$ if:

- For all pairs $(t_1, t_2)$ of tuples in relation $R$ such that $t_1[A] = t_2[A]$, it is also the case $t_1[B] = t_2[B]$.

Note: let $K$ be a super-key of $R$, then $K \rightarrow R$.

**Examples**

**Emp:**
- $emp \rightarrow ename$
- $emp \rightarrow dno$

**Dept:**
- $dno \rightarrow mgr$
- $dno \rightarrow floor$

**EO:**
- $dno \rightarrow floor$
- $emp \rightarrow ss$
- $emp \rightarrow mgr$
Note, a FD is a property of a particular relation schema, and not of its instance.

Example

<table>
<thead>
<tr>
<th>enue</th>
<th>enue</th>
<th>byr</th>
<th>Stl</th>
<th>duv</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Mike</td>
<td>60</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>Sally</td>
<td>55</td>
<td>85</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>Sara</td>
<td>65</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>Julie</td>
<td>70</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>John</td>
<td>48</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>Alex</td>
<td>70</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>Mike</td>
<td>60</td>
<td>45</td>
<td>1</td>
</tr>
</tbody>
</table>

It looks like

\[
\text{enue} \rightarrow \text{byr} \\
\text{enue} \rightarrow \text{duv}
\]

But this is not right. For example, I could add tuple

23 Mike 49 22 6
Various types of FD's help identify bad designs:

- **Trivial Dependency**: \((A, B) \rightarrow A\) (Identity)

- **Partial Dependency**: \((A, B)\) is a key and \(A \rightarrow C\)
  
  Ex: supply \((540, puo, proj40, scity, project, qty)\)
  Key is \((540, puo, proj40)\)

  \[\begin{align*}
  540 & \rightarrow \text{scity} \\
  \text{proj40} & \rightarrow \text{project}
  \end{align*}\]  
  partial dependencies

- **Transitive Dependency**: \(A\) is a key, \(B\) is not, and \(A \rightarrow B \rightarrow C\)

  Ex: ED(eno, ehr, dna, wgr) where eno is the key.

  \[\begin{align*}
  \text{eno} & \rightarrow \text{dna} \rightarrow \text{wgr} \\
  \text{eno} & \rightarrow \text{wgr}
  \end{align*}\]  
  transitive dependency

Depending on the existence of various FDs, schemes are classified in various Normal Forms.
INF: Every attribute has a atomic value (as opposed to set value) value. All relational model schemas are by definition INF.

2NF: INF and no partial dependencies.

3NF: 2NF and no transitive dependencies to attributes that are not part of a key. Equivalently, if \( X \rightarrow A \) is a FD then either:
- a) it is trivial, or
- b) \( X \) is a superkey, or
- c) \( A \) is a subset of a candidate key.

BCNF: (Boyce-Codd Normal Form) 3NF and no transitive dependencies. Equivalently, if \( X \rightarrow A \) is a FD then either:
- a) it is trivial, or
- b) \( X \) is a superkey.

4NF (5,6,...) Multivalued dependencies (rarely used, usually just theory for theory sake)

Distinguish Example Schemes

INF but not 2NF: Supply(sno, pno, order, qty, order, qty, qty, qty, qty, qty)

Key = (sno, pno, order)

2NF but not 3NF: ED( eno, ename, byr, sal, do, division, floor, mgr)

Key = (eno)
One receipt per person, food pair

3NF but not BCNF: Restaurant (person, food, rec_number)

Key = (person, food)

This is 3NF because it has transitive dependency to an attribute that is part of the key.

Each normal form is included in the next one higher up:

Ideally, a decomposer should satisfy:
- BCNF
- Lossless joins
- Dependency preservation

If we can not get this, we settle for:
- 3NF
- Lossless joins
- Dependency preservation

3NF example: name, floor, qsc, RA1id

Result is not 17
**Lossless joins:** The decomposition should be done in a way such that no information is lost.

**Example:**
\[ \text{Baron} = (\text{bruce, lorn, charne, quount}) \]

\[ \downarrow \text{decompose} \]

**Att.-scheme:**
- \( (\text{quount, charne}) \)

**Loan-scheme:**
- \( (\text{bruce, lorn, quount}) \)

<table>
<thead>
<tr>
<th>bruce</th>
<th>lorn</th>
<th>quount</th>
</tr>
</thead>
<tbody>
<tr>
<td>hampton</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>norfolk</td>
<td>2</td>
<td>1500</td>
</tr>
<tr>
<td>lorn</td>
<td>3</td>
<td>2000</td>
</tr>
<tr>
<td>NN</td>
<td>4</td>
<td>1500</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>quount</th>
<th>charne</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Sue</td>
</tr>
<tr>
<td>1500</td>
<td>Mike</td>
</tr>
<tr>
<td>2000</td>
<td>Biff</td>
</tr>
<tr>
<td>1500</td>
<td>Jane</td>
</tr>
</tbody>
</table>

\( \text{aut} \times \text{lorn} \)

\[ \begin{array}{c|c|c|l} 
\text{bruce} & \text{lorn} & \text{quount} & \text{charne} \\
\hline 
\text{hampton} & 1 & 1000 & \text{Sue} \\
\text{norfolk} & 2 & 1500 & \text{Mike} \\
\text{norfolk} & 2 & 1500 & \text{Jane} \\
\text{lorn} & 3 & 2000 & \text{Biff} \\
\text{NN} & 4 & 1500 & \text{Mike} \\
\text{NN} & 4 & 1500 & \text{Jane} \\
\end{array} \]

**Gives us incorrect results! Hence we have lost information.**
Let \( F = \text{Set of all semi-trivial FDs} \)

Move FDs can be inferred from \( F \)

\[ F^+ = \text{Set of all inferred FDs} \]

Example:

\[ F = \{ A \rightarrow B, A \rightarrow C, B \rightarrow K \} \]

Then,

\[ F^+ = \{ A \rightarrow B, A \rightarrow C, A \rightarrow K, B \rightarrow K \} \]

The closure of \( F \) can be systematically determined using the inference rules:

(Note: \( F \vdash \{ x \rightarrow y \} \) denotes that \( x \rightarrow y \) can be inferred from \( F \)).

1. Reflexive rule: If \( x \subseteq Y \), then \( x \rightarrow y \)
2. Augmentation rule:
   - a) \( \{ x \rightarrow Y \} \vdash xZ \rightarrow YZ \)
   - b) \( \{ x \rightarrow Y \} \vdash xZ \rightarrow Y \)
3. Transitive rule: \( \{ x \rightarrow Y, y \rightarrow Z \} \vdash x \rightarrow Z \)
4. Decomposition rule: \( \{ x \rightarrow YZ \} \vdash x \rightarrow Y \)
5. Additive rule: \( \{ x \rightarrow Y, x \rightarrow Z \} \vdash x \rightarrow YZ \)
6. Pseudotransitive rule: \( \{ x \rightarrow Y, y \rightarrow Z \} \vdash x \rightarrow Z \)
When decomposing a schema we must make sure there are no lossless joins. This can be done as follows:

Let \( R \) be a schema we are to decompose. \( R_1 + R_2 \) be the decomposed schema. (Let \( R_1 \cup R_2 = R \))

The decomposition is a lossless-join decomposition if one of the two is true:

1. \( R_1 \cap R_2 \rightarrow R_1 \)
2. \( R_1 \cap R_2 \rightarrow R_2 \)

Note

\[ \text{Aut-scheme} \land \text{Lom-scheme} = \text{equivalent} \]

\[ \text{equivalent} \rightarrow \text{Aut-scheme} \]

\[ \text{equivalent} \rightarrow \text{Lom-scheme} \]

Hence, decompositions may lose information if is rejected.

**Dependency Preservation**

The goal is to preserve functional dependencies without requiring a join. (The purpose of DBMS checks whether or not the inserts, deletes, modifications.

- First, let \( F = \{(S, T, R) \mid \text{of all functional dependencies in } R \} \)

- The closure of \( F \), denoted by \( F^+ \), is the set of all logically implied dependencies.

\[
\begin{align*}
F &= \{(A \rightarrow B), (A \rightarrow C), (B \rightarrow H)\}, & F^+ &= \{(A \rightarrow B), (A \rightarrow C), (A \rightarrow H), (B \rightarrow H)\}
\end{align*}
\]
(1), (2), and (3) are known as Armstrong's axioms.

- Armstrong axioms are complete, i.e. by repeatedly applying (1) \( F^+ \) can be determined.

Algorithm for determining closure of attribute set \( \delta \):

\[
\text{result} = \delta \\
\text{while (changes to result)} \\
\begin{align*}
&\text{for each FD} \quad \beta \rightarrow \gamma \in F \\
&\text{if } \beta \subseteq \text{result} \\
&\quad \text{then } \text{result} = \text{result} \cup \gamma \\
\end{align*}
\]

Example:

Assume \( F = \{ A \rightarrow B, A \rightarrow C, C \rightarrow H, C \rightarrow I, B \rightarrow H \} \)

\( A^+ \) (the closure of \( A \) under \( F \)) can be calculated:

\[
\begin{array}{c|c|c}
\text{level} & \text{result} \\
0 & A \\
1 & A, B \quad A \rightarrow B \\
1 & A, B, C \quad C \rightarrow H, C \rightarrow I, B \rightarrow H \\
1 & A, B, C, H \quad \text{thus } A^+ = \{ A, B, C, H \} \\
\end{array}
\]
Har about \( \{AG\}^+ \)?

I do not know.

Note \( F^+ = \{A, B, C, G, H, I\}^+ \)

In worst case this alphabet may be the true attribute in the size of \( F \). There is a linear (but non-complex) algorithm.

- Often \( F_D \) are used to make sure the DB does not move into an incorrect state by the insertion of a fake or delete.

- In order to minimize the number of \( F_D \) that need to be tested, restrict \( F \) into a smaller (but just as meaningful) set.
  1. combine several \( F_D \) into one where possible
  2. remove extraneous attributes

- Attribute \( A \) is extraneous in \( \alpha \) if \( A \in \alpha \) and \( F \) logically implies \( (F - 3D \rightarrow B) \cup 3 (\alpha - A) \rightarrow B \)

- Attribute \( A \) is extraneous in \( \beta \) (\( \alpha \rightarrow \beta \)) if \( A \in \beta \) and \( F \) logically implies \( (F - 3D \rightarrow B) \cup 3D \rightarrow (\beta - A) \)
A canonical cover (or minimal cover) $F_c$
is a set of FD such that $F$ logically
implies all FD in $F_c$, and $F_c$ logically
implies all FD in $F$. Furthermore, $F_c$ must
have the following properties:
(i) no FD in $F_c$ contain an extraneous attribute
(ii) every FD has a single value on the RHS.
(iii) we can not remove a FD and still be equivalent to $F$.

Example: calculate $F_c$ for $F =$

\[ A \rightarrow BC \]
\[ B \rightarrow C \]
\[ A \rightarrow B \]
\[ AB \rightarrow C \]

1) Write RHS true single value only
\[ F = \begin{align*}
A \rightarrow B \\
A \rightarrow C \\
B \rightarrow C
\end{align*} \]

2) $A$ is extraneous in $AB \rightarrow C$ since $B \rightarrow C$ logically implies $AB \rightarrow C$.

Thus, $(\{A \rightarrow C\} \cup \{B \rightarrow C\})$ logically implies $F^+$.

Removing $A$ from $AB \rightarrow C$ gives $B \rightarrow C$ which is

Liu and result
\[ F^+ \Rightarrow \{ A \rightarrow B \}
\]

3) We can remove $A \rightarrow C$ and still be equivalent to $F$

\[ F^+ \Rightarrow \{ A \rightarrow B \} \Rightarrow F^+ \]

\[ F_c = \{ A \rightarrow B \} \]

\[ \{ B \rightarrow C \} \]
Minimal Cover

Formal Algorithm:

1. \( s \) \( \mathcal{G} = \mathcal{F} \)
2. replace each \( FD \) \( x \rightarrow A_1, A_2, \ldots, A_n \) in \( \mathcal{G} \) by \( FDs \) \( x \rightarrow A_1, x \rightarrow A_2, \ldots, x \rightarrow A_n \)
3. \((4)\) for each \( FD \) \( x \rightarrow A \) in \( \mathcal{G} \)
   - compute \( x^+ \) with respect to \( \mathcal{G} - \{ x \rightarrow A \} \)
   - if \( x^+ \) contains \( A \), remove \( x \rightarrow A \) for \( \mathcal{G} \)
4. \((5)\) for each remaining \( FD \) \( x \rightarrow A \) in \( \mathcal{G} \)
   - for each attribute \( B \) in \( x \) not an element of \( x \)
     - compute \( (x-B)^+ \) with respect to \( \mathcal{G} - \{ x \rightarrow A \} \)
     - if \( (x-B)^+ \) contains \( A \), replace \( x \rightarrow A \) with \( (x-B) \rightarrow A \) in \( \mathcal{G} \)

These methods can be used to determine if a relation schema decomposition is dependency preserving.
Minimal Cover Example

\[ F = \{ A \rightarrow B \rightarrow C, F \rightarrow A, E \rightarrow F, G \rightarrow C, H \rightarrow F \} \]

**STEP 1**

\( A \rightarrow B \rightarrow C \)

\( A \rightarrow D \)

\( C \rightarrow E \)

\( B \rightarrow G \rightarrow F \)

**STEP 2**

1. Does \((A \rightarrow B)^* \in C\) if \(G\) is removed?
   - No → can remove \(G\)
2. Does \(A^+ \in C\) if removed \(G\) → No
3. Does \(C^+ \in E\) if removed \(G\) → No
4. Does \((B \rightarrow G)^+ \in E\) if removed \(G\)?
   - Yes, can remove \(C\)
5. \(F^+ \in A\) if removed \(G\)? No
6. \(F^+ \in B\) → Can remove \(F\)
7. No
8. No
STEP 2: Venue: Nunnari ATTR

1. Is A extreem in $A0H \rightarrow C$?
   
   Yes, if $A0H \rightarrow C$. Then derive $E$ from $E = (A0H \rightarrow C) \land (B0H \rightarrow C)$. Further:
   
   - $B0H \rightarrow E$
   - $E \rightarrow F \Rightarrow B0H \rightarrow A$
   - $F \rightarrow AO \Rightarrow A0H \rightarrow C$

2. Is B extreem in $B0H \rightarrow C$? NO

3. Is H extreem in $B0H \rightarrow C$? NO

4. Is G extreem in $B6H \rightarrow F$?
\[ BH \rightarrow E \]
\[ E \rightarrow F \]

\[ \text{thus } BGH \rightarrow F, \text{ e.g. G is } \]
\[ \text{ext} \]
\[ \Rightarrow BH \rightarrow F \]

\textit{End of Step 2}

- BH \rightarrow C
- A \rightarrow D
- C \rightarrow E
- DH \rightarrow F
- F \rightarrow A
- E \rightarrow F

\textit{Initial cover}

\text{Step 3}

- C \rightarrow E
- A \rightarrow D

\text{remove}

\text{redundant}

F \rightarrow A
E \rightarrow F

BH \rightarrow E
No, how about schema decomposition:

Let $R$ be decomposed into $R_1, R_2, \ldots, R_n$.

The restriction of $F$ to $R_i$ is the set $F_i$ of all functional dependencies that include only attributes of $R_i$.

Let $F' = F_1 \cup F_2 \cup \ldots \cup F_n$.

If $F'^+ = F^+$ the decomposition is dependency preserving.

Example of how this is used:

**Banker-Schema**: $(\text{branch-name}, \text{customer-name}, \text{banker-name})$

$F = \{\text{banker} \rightarrow \text{branch}, \text{customer, branch} \rightarrow \text{banker}\}$

Note, key = $(\text{customer, branch})$

**Banker-Schema is not BCNF, why?**

banker $\rightarrow$ branch, banker is not a super key, dependency is non-trivial.

\[ \downarrow \text{decompose} \]

Banker-Branch-Schema $(\text{banker, branch})$

Customer-Banker-Schema $(\text{customer, banker})$

**This is BCNF + lossless, but does it preserve dependency?**
\[ F_1 = \{ \text{baker} \rightarrow \text{bunch} \} \]
\[ F_2 = \emptyset \quad \text{(no trans. dependencies)} \]
\[ \Rightarrow F'^+ \neq F^+ \]

Thus, we could not satisfy
a) BCNF, and
b) lossless, and
c) dependency preserving

But, we can get 3NF, lossless, dependency-preserving since

\[ \text{Baker-Scheme is 3NF} \]

\[ \text{Customer, bunch, baker} \]

1. \( X \rightarrow B \), \( X \) is a superkey
2. \( B \rightarrow A \), \( A \) is a subset of a candidate key

Advantage of Normal Form approach:
→ It is a way to formalize a process that is usually only done using intuition.
1) Try to decompose R into BCNF (lossless)

Algorithm:

1) set $D = E(R)$
2) while $\exists$ schema $G \in D$ that is not BCNF
   
   a) choose $G \in D$ that is not BCNF
   b) find FD $X \rightarrow Y$ that violates BCNF
   c) replace $G$ by the schemas $(G - Y) \cup (xU)$

Note: the final decomposition may not be lossless dependency preserving.

2) => test to see if dependency preserving

3) If not dependency preserving, decompose as lossless & dependency preserving 3NF

Algorithm:

1) find minimal cover $F_c$ for $F$
2) for each left-hand side $X \in F_c$
   
   create a relation schema $(xU_A, U_A, U \ldots A_n)$
   
   where $X \rightarrow A_1, X \rightarrow A_2, \ldots X \rightarrow A_n$ are all $F_c$
   
   dependencies in $F_c$ with $X$ in left-hand side
3) replace all remaining (unplaced) attributes in
   
   a single relation
4) if one of the relation schemas contains k
   
   key of R, create one more relation schema
   
   that contains k attributes that form a new
Example:

\[
\text{lots} (\text{property}, \text{country}, \text{name}, \text{lot#}, \text{area}, \text{price}, \text{tax})
\]

\[\downarrow \text{abbreviate}\]

\[
\text{lots} (\text{pid}, \text{country}, \text{lot#}, \text{area}, \text{price}, \text{tax})
\]

\[
F = \left\{ \begin{array}{c}
\text{pid} \rightarrow \text{country, lot#}, \text{area}, \text{price}, \text{tax} \\
\text{country, lot#} \rightarrow \text{pid}, \text{area}, \text{price}, \text{tax} \\
\text{country} \rightarrow \text{tax} \\
\text{area} \rightarrow \text{price}
\end{array} \right\}
\]

\[\text{Note: } F^+ = F\]

Is lots BCNF?

No: \(\text{area} \rightarrow \text{price}\) is a partial dependency, not even 3NF!

Decompose (BCNF decomposition algorithm)

lots1 (\text{pid, country, lot#}, \text{area}, \text{price})
lots2 (\text{country, tax})

Are both of these BCNF?

No: \(\text{area} \rightarrow \text{price}\) is part of a transitive dependency!

\[
\begin{align*}
\text{lots1} & (\text{pid, country, lot#}, \text{area}) \\
\text{lots2} & (\text{country, tax}) \\
\text{BCNF?} & \text{ Yes!}
\end{align*}
\]
But is it dependent preserving?

\[ \begin{align*}
F_{\pi} &= \{ \text{pid} \to \text{curnc}, \text{lot#}, \text{quer} \} \\
F_{\Sigma} &= \{ \text{curnc}, \text{lot#} \to \text{pid}, \text{quer} \}
\end{align*} \]

\[ F_{j} = \{ \text{curnc} \to \text{tax} \} \]

\[ F_{1}^{+} = \begin{cases}
\text{pid} \to \text{curnc}, \text{lot#}, \text{quer} \\
\text{curnc}, \text{lot#} \to \text{pid}, \text{quer} \\
\text{quer} \to \text{price} \\
\text{curnc} \to \text{tax}
\end{cases} + 
\]

\[ = F_{1}^{+} \]

\[ \text{So, Yes! It is dependent preserving.} \]
Let's add a 3rd FD to our example:

\[ \text{avc} \rightarrow \text{curnc} \]

In this case, lets say is need BCF

\[ \text{avc, poa} \]

1. lots1x (pid, avc, lot#) ⊆ lotsby (avc, curnc)
2. lots1b (avc, price) ⊆ lots2 (curnc, tax)

Is it BCF? Yes

Is it dependent preserving?

\[ F_1 = \text{pid} \rightarrow \text{avc}, \text{lot#} \]
\[ F_2 = \text{avc} \rightarrow \text{curnc} \]
\[ F_3 = \text{avc} \rightarrow \text{price} \]
\[ F_4 = \text{avc} \rightarrow \text{tax} \]

\[ F_1^+ = \{ \text{pid} \rightarrow \text{curnc}, \text{lot#}, \text{avc}, \text{price}, \text{tax} \}
\]
\[ \text{curnc} \rightarrow \text{tax} 
\]
\[ \text{avc} \rightarrow \text{price}, \text{curnc}, \text{tax} \]

but NOT curnc, lot# → pid, avc, price, tax!

\[ \text{Not Dmitry Preservin} \]
Thus, we must settle for IMF, lacking presence, tasteless and here use list decomposition algorithm:

$$F = \begin{cases} 
\text{pid} \rightarrow \text{crime, lot#, area, price, tax} \\
\text{crime, lot#} \rightarrow \text{pid, area, price, tax} \\
\text{crime} \rightarrow \text{tax} \\
\text{area} \rightarrow \text{price} \\
\text{area} \rightarrow \text{crime} 
\end{cases}$$

Find limited cover:

\begin{align*}
1 & \text{ pid } \rightarrow \text{crime} \\
2 & \text{ " } \rightarrow \text{lot#} \\
3 & \text{ " } \rightarrow \text{area} \\
4 & \text{ " } \rightarrow \text{price} \\
5 & \text{ " } \rightarrow \text{tax} \\
6 & \text{crime, lot#} \rightarrow \text{pid} \\
7 & \text{ " } \rightarrow \text{area} \\
8 & \text{ " } \rightarrow \text{price} \\
9 & \text{ " } \rightarrow \text{tax} \\
10 & \text{area } \rightarrow \text{price} \\
11 & \text{area } \rightarrow \text{crime} \\
12 & \text{crime} \rightarrow \text{tax} \\
\end{align*}

\begin{align*}
\text{pid } & \Rightarrow \text{crime, lot#, area, price, tax} \\
\text{this includes crime, hence can remove 1} \\
\text{can not remove 2} \\
\text{hct 7} \\
\text{yes 4} \\
\text{yes 5} \\
\text{hct 6} \\
\text{yes 7} \\
\text{yes 8} \\
\text{crime, lot# } \rightarrow \text{pid} \\
\text{yes 5} \\
\text{hct 10} \\
\text{hct 11} \\
\text{hct 12} \\
\end{align*}
\[ F^c = \text{minimal cover} = \{ \begin{align*} \text{pid} &\rightarrow \#_1 \\
\text{pid} &\rightarrow \text{area} \\
(\text{name, \#_1}) &\rightarrow \text{pid} \\
\text{area} &\rightarrow \text{price} \\
\text{area} &\rightarrow \text{name} \\
\text{name} &\rightarrow \text{tax} \end{align*} \] 

BNF: lessless, presence code control:

\( \gamma (\text{pid, \#_1, area}) \quad \gamma (\text{area, price, name}) \)

\( \gamma (\text{name, \#_1, pid}) \quad \gamma (\text{name} \rightarrow \text{tax}) \)

It is 2NF:

(marked as OCMF because \( \text{name, \#_1} \rightarrow \text{pid} \))
Check if it is depending presently:

\[ F_1 = \text{pid} \rightarrow \text{1#}, \text{area} \]
\[ F_2 = \text{area} \rightarrow \text{price}, \text{name} \]
\[ F_3 = \text{name,1#} \rightarrow \text{pid} \]
\[ F_4 = \text{area} \rightarrow \text{tax} \]

\[ F^+ = \begin{cases} \text{pid} \rightarrow \text{1#}, \text{area}, \text{price}, \text{name}, \text{tax} \\ \text{name,1#} \rightarrow \text{pid}, \text{area}, \text{price}, \text{tax} \\ \text{area} \rightarrow \text{tax} \\ \text{area} \rightarrow \text{price}, \text{name}, \text{tax} \end{cases} \]