Design Theory for Relational Databases

- Functional Dependencies
- Decompositions
- Normal Forms: BCNF, Third Normal Form
- Introduction to Multivalued Dependencies
A poor choice of a relational database schema can lead to redundancy and related anomalies.

We’ve seen the E/R approach for overall database schema design.

Here we want to look at relation schemas and
- Consider problems that might arise with a poor choice of schema,
- Evaluate a schema with regards to redundancy and other anomalies, and
- Determine how to come up with a better design by decomposing a relational schema.

Key notion: functional dependency.
Functional Dependencies

- Functional dependencies generalize the notion of a key of a relation.
- Write a FD as $X \rightarrow Y$ where $X$ and $Y$ are sets of attributes.
- $X \rightarrow Y$ is an assertion about a relation $R$ that whenever two tuples of $R$ agree on all the attributes of $X$, then they must also agree on all attributes in set $Y$.
  - Say “$X \rightarrow Y$ holds in $R$.”
  - Convention: $\ldots$, $X$, $Y$, $Z$ represent sets of attributes; $A$, $B$, $C$, $\ldots$ represent single attributes.
  - Convention: No set notation for sets of attributes; use $ABC$, rather than $\{A,B,C\}$. 
Example: FD’s

Customers(name, addr, beersLiked, manf, favBeer)

Reasonable FD’s to assert:

1. name -> addr favBeer
   - Note this FD is the same as name -> addr and name -> favBeer.
2. beersLiked -> manf
<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
<th>manf</th>
<th>favBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Export</td>
<td>Molson</td>
<td>G.I. Lager</td>
</tr>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>G.I. Lager</td>
<td>Gr. Is.</td>
<td>G.I. Lager</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Export</td>
<td>Molson</td>
<td>Export</td>
</tr>
</tbody>
</table>

Because `name -> addr`

Because `name -> favBeer`

Because `beersLiked -> manf`
Splitting Right Sides of FD’s

- $X \rightarrow A_1 A_2 \ldots A_n$ holds for $R$ exactly when each of $X \rightarrow A_1$, $X \rightarrow A_2$, ..., $X \rightarrow A_n$ hold for $R$.
- **Example:** $A \rightarrow BC$ is equivalent to $A \rightarrow B$ and $A \rightarrow C$.
- There is no splitting rule for left sides.
- We’ll generally express FD’s with singleton right sides.
Keys of Relations

- Let $K$ be a set of attributes (possibly singleton) in a relation $R$.
- $K$ is a **superkey** for relation $R$ if $K$ functionally determines all attributes of $R$.
- $K$ is a **key** for $R$ if $K$ is a superkey, but no proper subset of $K$ is a superkey.
  - Also called a **candidate key**
- A **primary key** is a candidate key that has been selected as the means of identifying tuples in a relation.
Example: Superkey

Customers(name, addr, beersLiked, manf, favBeer)

- \{name, beersLiked\} is a superkey because together these attributes determine all the other attributes.
  - name -> addr favBeer
  - beersLiked -> manf
Example: Key

■ \{\text{name, beersLiked}\} is a key because neither \{\text{name}\} nor \{\text{beersLiked}\} is a superkey.
  ● \text{name} doesn’t \rightarrow \text{manf}; \text{beersLiked} doesn’t \rightarrow \text{addr}.

■ There are no other keys, but lots of superkeys.
  ● Any superset of \{\text{name, beersLiked}\} is a superkey.
Where Do Keys Come From?

1. Just assert a key $K$
   - E.g. student number
   - Have FD’s $K \rightarrow A$ for all attributes $A$.
2. Determine FD’s and deduce the keys by systematic exploration.
More FD’s From “Physics”

- Example: “no two courses can meet in the same room at the same time” tells us: hour room -> course.
  - i.e. commonsense constraints
We are given FD’s

\[ X_1 \rightarrow A_1, \; X_2 \rightarrow A_2, \; \ldots, \; X_n \rightarrow A_n, \]

and we want to know whether an FD \( Y \rightarrow B \) must hold in any relation that satisfies the given FD’s.

- Example: If \( A \rightarrow B \) and \( B \rightarrow C \) hold, surely \( A \rightarrow C \) holds, even if we don’t say so.

- Important for design of good relation schemas.
To test if $Y \rightarrow B$ holds, given a set of FDs, start by assuming that two tuples agree in all attributes of $Y$.

$\leftarrow Y \rightarrow$

\[
\begin{align*}
a_1 & a_2 a_3 b_1 b_2. \ldots \\
a_1 & a_2 a_3 c_1 c_2. \ldots
\end{align*}
\]

Use the given FD’s to infer that these tuples must also agree in certain other attributes.

- If $B$ is one of these attributes, then $Y \rightarrow B$ is true.
- Otherwise, the two tuples, with any forced equalities, form a two-tuple relation that proves $Y \rightarrow B$ does not follow from the given FD’s.
An easier way to test is to compute the *closure* of $Y$, denoted $Y^+$. 

- **Basis**: $Y^+ = Y$.

- **Induction**: Look for a FD whose left side $X$ is a subset of the current $Y^+$.
  - If the FD is $X \rightarrow A$, add $A$ to $Y^+$. 
Diagrammatically:

Given $Y^+$ and $X \rightarrow A$: 

- $X$  
- $A$  
- $Y^+$  
- new $Y^+$
Finding All Implied FD’s

- Sometimes, for a relation R with a set of FD’s, we want to find those FD’s that hold in subrelations of R.

- **Motivation:** “normalization,” the process where we break a relation schema into two or more schemas for better performance
  - More on normalization later...

- Example: $ABCD$ with FD’s $AB \rightarrow C$, $C \rightarrow D$, and $D \rightarrow A$.
  - Decide to decompose into $ABC$, $AD$.
  - Ask: what FD’s hold in $ABC$?
  - Answer: not only $AB \rightarrow C$, but also $C \rightarrow A$!
ABCD: \( a_1b_1cd_1 \quad a_2b_2cd_2 \)  
\( d_1 = d_2 \) because \( C \rightarrow D \)  
\( a_1 = a_2 \) because \( D \rightarrow A \)
Thus, tuples in the projection with equal C’s have equal A’s. Hence $C \rightarrow A$. 
1. Start with given FD’s and find all nontrivial FD’s that follow from the given FD’s.
   - Nontrivial = right side not contained in the left.
2. Select those FD’s that involve only attributes of the projected schema.
1. For each set of attributes $X$, compute $X^+$. 
2. Add $X \rightarrow A$ for all $A$ in $X^+ - X$. 
3. However, drop $XY \rightarrow A$ whenever we discover $X \rightarrow A$. 
   - Because $XY \rightarrow A$ follows from $X \rightarrow A$. 
4. Finally, use only FD’s involving projected attributes.
A Few Tricks

- Trivially, there’s no need to compute the closure of the empty set or of the set of all attributes.
- If we find $X^+ = \text{all attributes}$, the closure of any superset of $X$ is also the set of all attributes.
  - So if $X^+ = \text{all attributes}$, you don’t need to consider any supersets of $X$. 
Example: Projecting FD’s

- **ABC** with FD’s **A ->B** and **B ->C**.

  Ask, what FDs hold on **AC**?

- Compute the closure.
  - **A^+ = ABC**; yields **A ->B, A ->C**.
    - We do not need to compute **(AB)^+** or **(AC)^+**.
  - **B^+ = BC**; yields **B ->C**.
  - **C^+ = C**; yields nothing new.
  - **(BC)^+ = BC**; yields nothing new.
  - Resulting FD’s: **A ->B, A ->C, and B ->C**.

- Projection onto **AC**: **A ->C**.
  - This is the only FD that involves \{A,C\}. 
Relational Schema Design

- We can use FD’s to help us design a good relational schema, given an existing database schema.

- Goal of relational schema design is to avoid anomalies and redundancy.
  - *Update anomaly*: one occurrence of a fact is changed, but not all occurrences.
  - *Deletion anomaly*: valid fact is lost when a tuple is deleted.

- Overall idea: Ensure that relations are in some normal form, which will guarantee that the relation has certain (good) properties.

- We’ll look at:
  - Boyce-Codd Normal Form (BCNF)
  - 3rd Normal Form
  - And briefly look at 4th Normal Form
Example of Bad Design

Customers(\textbf{name, addr, beersLiked, manf, favBeer})

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
<th>manf</th>
<th>favBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Export</td>
<td>Molson</td>
<td>G.I. Lager</td>
</tr>
<tr>
<td>Janeway</td>
<td>???</td>
<td>G.I. Lager</td>
<td>Gr. Is.</td>
<td>???</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Export</td>
<td>Export</td>
<td>Export</td>
</tr>
</tbody>
</table>

Data is redundant, because each of the ???’s can be figured out by using the FD’s name -> addr favBeer and beersLiked -> manf.
This Bad Design Also Exhibits Anomalies

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
<th>manf</th>
<th>favBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Export</td>
<td>Molson</td>
<td>G.I. Lager</td>
</tr>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>G.I. Lager</td>
<td>Gr. Is.</td>
<td>G.I. Lager</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Export</td>
<td>Molson</td>
<td>Export</td>
</tr>
</tbody>
</table>

- **Update anomaly**: If Janeway is transferred to Intrepid, we have to remember to change each of her tuples.
- **Deletion anomaly**: If nobody likes Export, we lose track of the fact that Molson manufactures Export.
We say a relation \( R \) is in BCNF if whenever \( X \rightarrow Y \) is a nontrivial FD that holds in \( R \), \( X \) is a superkey.

Remember:
- **nontrivial** means \( Y \) is not contained in \( X \).
- **superkey** is any superset of a key (not necessarily a proper superset).
Example

Customers(name, addr, beersLiked, manf, favBeer)
FD’s: name->addr favBeer, beersLiked->manf

- The only key is \{name, beersLiked\}.
- In each FD, the left side is not a superkey.
- Either of these FD’s shows that Customers is not in BCNF
Another Example

Beers(name, manf, manfAddr)

FD’s: name->manf, manf->manfAddr

- The only key is \{name\}.  
- name->manf does not violate BCNF, but manf->manfAddr does.
**Goal**: For relation R not in BCNF, decompose R into subrelations that are in BCNF.

**Given**: Relation R with FD’s F.

Look among the given FD’s for a BCNF violation $X \rightarrow Y$.

- If any FD following from F violates BCNF, then there will surely be an FD in F itself that violates BCNF.

**Compute** $X^+$.

- We won’t get all attributes, since X isn’t a superkey.
Decompose $R$ using $X \rightarrow Y$

- **Replace** $R$ by relations with schemas:
  1. $R_1 = X^+$.
  2. $R_2 = R - (X^+ - X)$.

- **Project** the given FD's $F$ onto the two new relations.
  - Recall that this requires finding the implicit FD's.
Decomposition Picture

\[ R - X^+ \quad X \quad X^+ - X \]

\[ R_1 \quad R_2 \quad R \]
Example: BCNF Decomposition

Customers(name, addr, beersLiked, manf, favBeer)

\[ F = \text{name->addr, name -> favBeer, beersLiked->manf} \]

- Pick BCNF violation name->addr.
- Close the left side: \{name\}+ = \{name, addr, favBeer\}.
- Decomposed relations:
  1. Customers1(name, addr, favBeer)
  2. Customers2(name, beersLiked, manf)
We are not done.

- We need to check Customers1 and Customers2 for BCNF violations.

Projecting FD’s is easy here.

For Customers1\((\text{name, addr, favBeer})\), relevant FD’s are name->addr and name->favBeer.

- Thus, \{name\} is the only key and Customers1 is in BCNF.
For Customers2(name, beersLiked, manf), the only FD is beersLiked->manf, and the only key is \{name, beersLiked\}.

Violation of BCNF.

beersLiked* = \{beersLiked, manf\}, so we decompose Customers2 into:

1. Customers3(beersLiked, manf)
2. Customers4(name, beersLiked)
Example -- Concluded

The resulting decomposition of Customers:
1. Customers1(name, addr, favBeer)
2. Customers3(beersLiked, manf)
3. Customers4(name, beersLiked)

Notice:
- Customers1 tells us about customers,
- Customers3 tells us about beers, and
- Customers4 tells us the relationship between customers and the beers they like.
There is one configuration of FD’s that causes trouble when we decompose.

- \( AB \rightarrow C \) and \( C \rightarrow B \).
  - Example: \( A \) = street address, \( B \) = city, \( C \) = postal code.
- There are two keys, \( \{A,B\} \) and \( \{A,C\} \).
- \( C \rightarrow B \) is a BCNF violation, so to get BCNF we decompose into \( AC, BC \).
We Cannot Enforce FD’s

- The problem is that if we use $AC$ and $BC$ as our database schema, we cannot enforce the FD $AB \rightarrow C$ by checking FD’s in these decomposed relations.

- Example with $A =$ street, $B =$ city, and $C =$ postal code on the next slide.
**An Unenforceable FD**

<table>
<thead>
<tr>
<th>street</th>
<th>p.c.</th>
<th>city</th>
<th>p.c.</th>
</tr>
</thead>
<tbody>
<tr>
<td>545 Tech Sq.</td>
<td>02138</td>
<td>Cambridge</td>
<td>02138</td>
</tr>
<tr>
<td>545 Tech Sq.</td>
<td>02139</td>
<td>Cambridge</td>
<td>02139</td>
</tr>
</tbody>
</table>

Join tuples with equal postal codes.

<table>
<thead>
<tr>
<th>street</th>
<th>city</th>
<th>p.c.</th>
</tr>
</thead>
<tbody>
<tr>
<td>545 Tech Sq.</td>
<td>Cambridge</td>
<td>02138</td>
</tr>
<tr>
<td>545 Tech Sq.</td>
<td>Cambridge</td>
<td>02139</td>
</tr>
</tbody>
</table>

Although no FD’s were violated in the decomposed relations, FD `street city -> p.c.` is violated by the database as a whole.
3rd Normal Form (3NF) modifies the BCNF condition so we do not have to decompose in this problem situation.

Define: An attribute is *prime* if it is a member of any key.

- $X \rightarrow A$ violates 3NF if and only if $X$ is not a superkey, and also $A$ is not prime.

- I.e. a relation $R$ is in third normal form just if for every nontrivial FD $X \rightarrow A$, either $X$ is a superkey or $A$ is prime (is a member of some key).
  - So this is weaker than BCNF.
  - I.e. if a relation is in BCNF then it must be in 3NF, but not necessarily vice versa.
Example: 3NF

- In our problem situation with FD’s \( AB \rightarrow C \) and \( C \rightarrow B \), we have keys \( AB \) and \( AC \).
- Thus \( A, B, \) and \( C \) are each prime.
- Although \( C \rightarrow B \) violates BCNF, it does not violate 3NF.
There are two important properties of a decomposition:

1. **Lossless Join**: If relation \( R \) is decomposed into \( R_1, R_2, \ldots, R_k \), it should be possible to reconstruct \( R \) exactly from \( R_1, R_2, \ldots, R_k \).
   
   I.e. in decomposing, no information in the original relation is lost.

2. **Dependency Preservation**: It should be possible to check in the projected relations whether all the original FD’s are satisfied.
   
   I.e. in decomposing, no information given in the functional dependencies is lost.
We can get (1) with a BCNF decomposition.

We can get both (1) and (2) with a 3NF decomposition.

But we can’t always get (1) and (2) with a BCNF decomposition.

- street/city/p.c. is an example.
Testing for a Lossless Join

- Ask: If we project $R$ onto $R_1, R_2, \ldots, R_k$, can we recover $R$ by rejoining?
- Clearly, any tuple in $R$ can be recovered from its projected fragments.
- So the only question is:
  
  When we rejoin, do we ever get back something we didn’t have originally?

- Below, projecting onto (AB) and (BC), and joining introduces tuples (1,2,1) and (3,2,3).
  
  Such a project/join is called lossy.
The Chase Test

- Assume $R$ is decomposed into $R_1, R_2, \ldots, R_k$
- Suppose tuple $t$ is in the join.
- Then $t$ is the join of projections of some tuples of $R$, one for each $R_i$ of the decomposition.
- Can we use the given FD’s to show that one of the tuples in the original relation must be $t$?
Start by assuming \( t = abc \ldots \) is in the join of the projected relations.

For each \( i \), there is a tuple \( s_i \) of \( R \) that has \( a, b, c, \ldots \) in the attributes of \( R_i \).

\( s_i \) can have any values in other attributes.

We’ll use the same letters as in \( t \), but with a subscript, for these components.
Example: The Chase

- Let $R = ABCD$, and the decomposition be $AB$, $BC$, and $CD$.
- Let the given FD’s be $C \rightarrow D$ and $B \rightarrow A$.
- Suppose the tuple $t = abcd$ is the join of tuples projected onto $AB$, $BC$, and $CD$.
- Then there are tuples $abc_1d_1$, $a_2bcd_2$, $abc_3d_3$ in $R$ that are projected and joined to give $abcd$. 
The Tableau

The tuples of R projected onto AB, BC, CD.

We’ve proved the second tuple must be $t$. 

Use $B \rightarrow A$ 

Use $C \rightarrow D$
Summary of the Chase

1. If two rows agree in the left side of a FD, make their right sides agree too.
2. Always replace a subscripted symbol by the corresponding unsubscripted one, if possible.
3. If we ever get an unsubscripted row, we know any tuple in the project-join is in the original (the join is lossless).
4. Otherwise, the final tableau is a counterexample.
Example: Lossy Join

- Same relation \( R = ABCD \) and same decomposition.
- But with only the FD \( C \rightarrow D \).
These three tuples are an example that shows that the join is lossy. *The tuple* \(abcd\) *is not in* \(R\), *but we can project and rejoin to get* \(abcd\).
3NF Synthesis Algorithm

- We can always construct a decomposition into 3NF relations with a lossless join and dependency preservation.
- Need *minimal basis* for the FD’s.
- A *basis* for a set of FD’s S is a set of FD’s that is equivalent to S.
- A *minimal basis* is a basis that satisfies the constraints:
  1. Right sides are single attributes.
  2. No attribute can be removed from a left side.
  3. No FD can be removed
- A minimal basis is also called a *canonical cover*. 
Constructing a Minimal Basis

1. Split right sides of FDs.

2. Repeatedly try to remove an attribute from a left side and see if the resulting FD’s are equivalent to the original.
   - I.e. A is redundant in AX->B wrt F if F implies X->B

3. Repeatedly try to remove an FD and see if the remaining FD’s are equivalent to the original.
   - I.e. X->A is redundant wrt F if F - {X->A} implies X->A

Aside: The steps need to be done in the above order.
3NF Synthesis – (2)

- Determine a minimal basis.
- Form one relation for each FD in the minimal basis.
  - Schema is the union of the left and right sides of each FD.
  - An optimisation: If there are >1 FDs in the minimal basis with the same LHS, they can be combined.
    - I.e. for X -> A, X -> B, can produce the relation XAB
- If none of the relations above is a superkey for the original relation R, then add one relation whose schema is a key for R.
Relation \( R = ABCD \).

FD’s \( A \rightarrow B \) and \( A \rightarrow C \).

These FDs form a minimal basis.

**Decomposition**: AB and AC from the FD’s, plus AD for a key.

- As noted, we can also first combine FDs with the same LHS.
- I.e. it’s better is to decompose to ABC and AD.
Another Example

- Relation R = ABC
- FD’s AB->C and C->B. (These form a minimal basis.)
- Strictly speaking we get ABC and BC.
- But it never makes sense to have a relation whose schema is a subset of another, so we drop BC.
Why the 3NF Synthesis Algorithm Works

- **Preserves dependencies**: Each FD from a minimal basis is contained in a relation, thus preserved.
- **Lossless Join**: Use the chase to show that the row for the relation that contains a key can be made all-unsubscripted variables.
- **3NF**: Hard part – a property of minimal bases.
Other Normal Forms

- First Normal Form says that attribute values are atomic.
- Second Normal Form is a restricted version of 3NF and is of historical interest only.
- Fourth Normal Form deals with *multivalued dependencies* (MVD’s).
Definition of MVD

- A multivalued dependency (MVD) on $R$ is an assertion that two attributes or sets of attributes, are independent of each other.

- The MVD $X \rightarrow\rightarrow Y$ says that if two tuples of $R$ agree on all the attributes of $X$, then their components in $Y$ may be swapped, and the result will be two tuples that are also in the relation.

- I.e., for each value of $X$, the values of $Y$ are independent of the values of $R-(X-Y)$. 
Example: MVD

Customers(name, addr, phones, beersLiked)

- A customer’s phones are independent of the beers they like.
  - name->->phones and name -->beersLiked.

- Thus, each of a customer’s phones appears with each of the beers they like in all combinations.

- This repetition is unlike FD redundancy.
  - name->addr is the only FD.

- In fact, note that Customers(name, phones, beersLiked) is in BCNF, though there potential redundancy due to the MVD.
Tuples Implied by name-\rightarrow\rightarrow\text{phones}

If we have tuples:

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>phones</th>
<th>beersLiked</th>
</tr>
</thead>
<tbody>
<tr>
<td>sue</td>
<td>a</td>
<td>p1</td>
<td>b1</td>
</tr>
<tr>
<td>sue</td>
<td>a</td>
<td>p2</td>
<td>b2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p2</td>
<td>b1</td>
</tr>
<tr>
<td>sue</td>
<td>a</td>
<td>p1</td>
<td>b2</td>
</tr>
</tbody>
</table>

Then these tuples must also be in the relation.
Fourth Normal Form

- The redundancy that comes from MVD’s is not removable by putting the database schema in BCNF.
- There is a stronger normal form, called 4NF, that (intuitively) treats MVD’s as FD’s when it comes to decomposition, but not when determining keys of the relation.
Decomposition and 4NF

- Decomposition into 4NF is much like BCNF decomposition
- A relation \( R \) is in 4NF if:
  - If \( X \rightarrow Y \) is a nontrivial MVD, then \( X \) is a superkey.
- \( X \rightarrow Y \) is a 4NF violation if
  - \( X \rightarrow Y \) is nontrivial but
  - \( X \) is not a superkey.
- If \( X \rightarrow Y \) is a 4NF violation for relation \( R \), we can decompose \( R \) using the same technique as for BCNF.
  1. \( XY \) is one of the decomposed relations.
  2. \( R - (X^+ - X) \) is the other.
We have focussed on BCNF and 3NF

Goals of a decomposition:
1. Elimination of anomalies
2. Recoverability of information (i.e. lossless join)
3. Preservation of dependencies

BCNF gives us 1 and 2 but not necessarily 3
3NF gives us 2 and 3 but not necessarily 1
There is no way to guarantee all three at the same time.
End: Design Theory for Relational Databases