Relational Database Design

or
“Troubles With Schemas“

Outline

- Relational Algebra (6.1)
- E/R Model (7.2 - 7.4)
- E/R Diagrams (7.5)
- Reduction to Schema (7.6)
- Relational Database Design (7.7)
- Functional Dependencies (8.1 – 8.4)
- Normalization (8.5 – 8.7)
- Relational Query Languages
- SQL Basics
- Formal Semantics of SQL
<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>inColor</th>
<th>StudioName</th>
<th>prodC#</th>
<th>StarName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Hamill</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Fisher</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>H. Ford</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
<td>187</td>
<td>Yes</td>
<td>Universal</td>
<td>150</td>
<td>Watts</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
<td>100</td>
<td>no</td>
<td>RKO</td>
<td>20</td>
<td>Fay</td>
</tr>
</tbody>
</table>

**Issues:**
1. Redundancy \(\rightarrow\) higher storage, inconsistencies ("anomalies")
   - update anomalies, insertion anomalies
2. Need nulls
   - Unable to represent some information without using nulls
   - How to store movies w/o actors (pre-productions etc)?

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>inColor</th>
<th>StudioName</th>
<th>prodC#</th>
<th>StarNames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>{Hamill, Fisher, H. Ford}</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
<td>187</td>
<td>Yes</td>
<td>Universal</td>
<td>150</td>
<td>Watts</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
<td>100</td>
<td>no</td>
<td>RKO</td>
<td>20</td>
<td>Fay</td>
</tr>
</tbody>
</table>

**Issues:**
3. Avoid sets
   - Hard to represent
   - Hard to query
Smaller schemas always good ????

Split Studio(name, address, presC#) into:
  Studio1 (name, presC#),
  Studio2(name, address)???

<table>
<thead>
<tr>
<th>Name</th>
<th>presC#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fox</td>
<td>101</td>
</tr>
<tr>
<td>Studio2</td>
<td>101</td>
</tr>
<tr>
<td>Universial</td>
<td>102</td>
</tr>
</tbody>
</table>

This process is also called “decomposition”

Issues:
4. Requires more joins (w/o any obvious benefits)
5. Hard to check for some dependencies
   What if the “address” is actually the presC#’s address ?
   No easy way to ensure that constraint (w/o a join).

Smaller schemas always good ????

Decompose StarsIn(movieTitle, movieYear, starName) into:
  StarsIn1(movieTitle, movieYear)
  StarsIn2(movieTitle, starName) ???

<table>
<thead>
<tr>
<th>movieTitle</th>
<th>movieYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>movieTitle</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>Hamill</td>
</tr>
<tr>
<td>King Kong</td>
<td>Watts</td>
</tr>
<tr>
<td>King Kong</td>
<td>Faye</td>
</tr>
</tbody>
</table>

Issues:
6. “joining” them back results in more tuples than what we started with
   (King Kong, 1933, Watts) & (King Kong, 2005, Faye)
   This is a “lossy” decomposition
   We lost some constraints/information
   The previous example was a “lossless” decomposition.
Desiderata

- No sets
- Correct and faithful to the original design
  - Avoid lossy decompositions
- As little redundancy as possible
  - To avoid potential anomalies
- No “inability to represent information”
  - Nulls shouldn’t be required to store information
- Dependency preservation
  - Should be possible to check for constraints

Not always possible. We sometimes relax these for: *simpler schemas, and fewer joins during queries.*
Outline

- Relational Algebra (6.1)
- E/R Model (7.2 - 7.4)
- E/R Diagrams (7.5)
- Reduction to Schema (7.6)
- Relational Database Design (7.7)
- Functional Dependencies (8.1 – 8.4)
- Normalization (8.5 – 8.7)
- Relational Query Languages
  - SQL Basics
  - Formal Semantics of SQL

Outline

- Mechanisms and definitions to work with FDs
  - Closures, candidate keys, canonical covers etc...
  - Armstrong axioms
- Decompositions
  - Loss-less decompositions, Dependency-preserving decompositions
- BCNF
  - How to achieve a BCNF schema
- BCNF may not preserve dependencies
- 3NF: Solves the above problem
- BCNF allows for redundancy
- 4NF: Solves the above problem
Approach

1. We will encode and list all our knowledge about the schema
   - Functional dependencies (FDs)
     - SSN $\rightarrow$ name (means: SSN “implies” (“determines”) length)
     - If two tuples have the same “SSN”, they must have the same “name”
     - movietitle $\rightarrow$ length  ???? Not true.
     - But, (movietitle, movieYear, movieDirector) $\rightarrow$ length --- True.
2. We will define a set of rules that the schema must follow to be considered good
   - “Normal forms”: 1NF, 2NF, 3NF, BCNF, 4NF, ...
   - A normal form specifies constraints on the schemas and FDs
3. If not in a “normal form”, we modify the schema

FDs: Example

<table>
<thead>
<tr>
<th>Course ID</th>
<th>Course Name</th>
<th>Dept Name</th>
<th>Credits</th>
<th>Section ID</th>
<th>Semester</th>
<th>Year</th>
<th>Building</th>
<th>Room No.</th>
<th>Capacity</th>
<th>Time Slot ID</th>
</tr>
</thead>
</table>

Functional dependencies:

- course_id $\rightarrow$ course_name, dept_name, credits
- building, room_number $\rightarrow$ capacity
- course_id, section_id, semester, year $\rightarrow$ building, room_number, time_slot_id
Let \( r(R) \) be a relation schema and 
\[ \alpha \subseteq R \text{ and } \beta \subseteq R \]

The functional dependency 
\[ \alpha \rightarrow \beta \]
holds on \( R \) iff for any legal relations \( r(R) \), whenever two tuples \( t_1 \) and \( t_2 \) of \( r \) have same values for \( \alpha \), they have same values for \( \beta \).

\[ t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta] \]

Example:

\[
\begin{array}{cc}
A & B \\
\hline
\end{array}
\]

On this instance, \( A \rightarrow B \) does NOT hold, but \( B \rightarrow A \) does hold.

---

**Functional Dependencies**

- **Difference between holding on an instance and holding on all legal relations**

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>inColor</th>
<th>StudioName</th>
<th>prodC#</th>
<th>StarName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Hamill</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Fisher</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>H. Ford</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
<td>100</td>
<td>no</td>
<td>RKO</td>
<td>20</td>
<td>Fay</td>
</tr>
</tbody>
</table>

- \( Title \rightarrow Year \) holds on this instance

- Is this a true functional dependency? **No.**
- Two movies in different years can have the same name.
- Can’t draw conclusions based on a single instance
  - Need **domain knowledge to decide which FDs hold**
FDs and Redundancy

- Consider a table: R(A, B, C):
  - With FDs: B → C, and A → BC
  - So “A” is a key, but “B” is not
- So: there is a FD whose left hand side is not a key
  - Leads to redundancy

Since B is not unique, it may be duplicated
Every time B is duplicated, so is C

Not a problem with A → BC
A can never be duplicated

Functional Dependencies

- Functional dependencies and keys
  - A key constraint is a specific form of a FD.
  - E.g. if α is a superkey for R, then:
    \[ \alpha \rightarrow R \]
  - Similarly for candidate keys and primary keys.

- Deriving FDs
  - A set of FDs may imply other FDs
  - e.g. If A → B, and B → C, then clearly A → C
  - We will see a formal method for inferring this later
Definitions

1. A relation instance $r$ satisfies a set of functional dependencies, $F$, if the FDs hold on the relation.

2. $F$ holds on a relation schema $R$ if no legal (allowable) relation instance of $R$ violates it.

3. A functional dependency, $\alpha \rightarrow \beta$, is called trivial if:
   - $\alpha$ is a superset of $\beta$
   - e.g. Movieyear, length $\rightarrow$ length

4. Given a set of functional dependencies, $F$, its closure, $F^+$, is all the FDs that are implied by FDs in $F$.

Approach

1. We will encode and list all our knowledge about the schema:
   - Functional dependencies (FDs)
   - Also:
     - Multi-valued dependencies (briefly discuss later)
     - Join dependencies etc...

2. We will define a set of rules that the schema must follow to be considered good:
   - “Normal forms”: 1NF, 2NF, 3NF, BCNF, 4NF, ...
   - A normal form specifies constraints on the schemas and FDs

3. If not in a “normal form”, we modify the schema.
A relation schema $R$ is “in BCNF” if:
- Every functional dependency $\alpha \rightarrow \beta$ that holds on it is EITHER:
  - 1. Trivial OR
  - 2. $\alpha$ is a superkey of $R$

Why is BCNF good?
- Guarantees that there can be no redundancy because of a functional dependency
- Consider a relation $r(A, B, C, D)$ with functional dependency
  - $A \rightarrow B$ and two tuples: $(a_1, b_1, c_1, d_1)$ and $(a_1, b_1, c_2, d_2)$
  - $b_1$ is repeated because of the functional dependency
  - BUT this relation is not in BCNF
  - $A \rightarrow B$ is neither trivial nor is $A$ a superkey for the relation

Why does redundancy arise?
- Given a FD, $\alpha \rightarrow \beta$, if $\alpha$ is repeated ($\beta \setminus \alpha$) has to be repeated
  1. If rule 1 is satisfied, ($\beta \setminus \alpha$) is empty, so not a problem.
  2. If rule 2 is satisfied, then $\alpha$ can’t be repeated, so this doesn’t happen either

Hence no redundancy because of FDs in BCNF
- Redundancy may exist because of other types of dependencies
  - Higher normal forms used for that (specifically, 4NF)
- Data may naturally have duplicated/redundant data
  - We can’t control that unless a FD or some other dependency is defined
Approach

1. We will encode and list all our knowledge about the schema:
   - Functional dependencies (FDs); Multi-valued dependencies; Join dependencies etc...

2. We will define rules the schema must follow to be “good”
   - “Normal forms”: 1NF, 2NF, 3NF, BCNF, 4NF, ...
   - A normal form specifies constraints on the schemas and FDs

3. If not in a “normal form”, we modify the schema
   - Through lossless decomposition (splitting)
   - Or direct construction using the dependencies information

----

BCNF

What if the schema is not in BCNF?
- Decompose (split) the schema into two pieces.

From the previous example: split the schema into:
- \( r_1(A, B), \ r_2(A, C, D) \)
- The first schema is in BCNF, the second one may not be (and may require further decomposition)
- No repetition now: \( r_1 \) contains \((a_1, b_1)\), but \( b_1 \) will not be repeated

Careful: you want the decomposition to be lossless
- No information should be lost
  - The above decomposition is lossless
- We will define this more formally later
Outline

- Mechanisms and definitions to work with FDs
  - Closures, candidate keys, canonical covers etc...
  - Armstrong axioms
- Decompositions
  - Loss-less decompositions, Dependency-preserving decompositions
- BCNF
  - How to achieve a BCNF schema
- BCNF may not preserve dependencies
- 3NF: Solves the above problem
- BCNF allows for redundancy
- 4NF: Solves the above problem

1. Closure

- Given a set of functional dependencies, $F$, its closure, $F^+$, is all FDs that are implied by FDs in $F$.
  - e.g. If $A \rightarrow B$, and $B \rightarrow C$, then clearly $A \rightarrow C$

- We can find $F^+$ by applying Armstrong’s Axioms:
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ (reflexivity)
  - if $\alpha \rightarrow \beta$, then $\gamma \alpha \rightarrow \gamma \beta$ (augmentation)
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (transitivity)

- These rules are
  - sound (generate only functional dependencies that actually hold)
  - complete (generate all functional dependencies that hold)
Additional rules

- If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$, then $\alpha \rightarrow \beta \gamma$ (union)
- If $\alpha \rightarrow \beta \gamma$, then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ (decomposition)
- If $\alpha \rightarrow \beta$ and $\gamma \beta \rightarrow \delta$, then $\alpha \gamma \rightarrow \delta$ (pseudotransitivity)

- The above rules can be inferred from Armstrong’s axioms.

Example

- $R = \{A, B, C, G, H, I\}$
- $F = \{A \rightarrow B$
  $\rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- Some members of $F^+$
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ and $B \rightarrow H$
  - $AG \rightarrow I$
    - by augmenting $A \rightarrow C$ with $G$, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI,$
    - and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI,$
    - and then transitivity
2. Closure of an attribute set

- Given a set of attributes \( \alpha \) and a set of FDs \( F \), closure of \( \alpha \) under \( F \) is the set of all attributes implied by \( \alpha \).

- In other words, the largest \( \beta \) such that: \( \alpha \rightarrow \beta \).

- Redefining super keys:
  - The closure of a super key is the entire relation schema.

- Redefining candidate keys:
  1. It is a super key.
  2. No subset of it is a super key.

Computing the closure for \( \alpha \)

- Simple algorithm.

  1. Start with \( \beta = \alpha \).
  2. Go over all functional dependencies, \( \delta \rightarrow \gamma \), in \( F^+ \).
  3. If \( \delta \subseteq \beta \), then
     - Add \( \gamma \) to \( \beta \).
  4. Repeat till \( \beta \) stops changing.
Example

- \( R = (A, B, C, G, H, I) \)
- \( F = \{ \begin{align*}
A & \rightarrow B \\
A & \rightarrow C \\
CG & \rightarrow H \\
CG & \rightarrow I \\
B & \rightarrow H
\end{align*} \} \)

- \( (AG)^+ \) ?
  - 1. result = AG
  - 2. result = ABCG \((A \rightarrow C \text{ and } A \rightarrow B)\)
  - 3. result = ABCG\text{H} \((CG \rightarrow H \text{ and } CG \subseteq AGBCH)\)
  - 4. result = ABCG\text{HI} \((CG \rightarrow I \text{ and } CG \subseteq AGBCH)\)

- Is \( (AG) \) a candidate key ?
  - 1. It is a super key.
  - 2. \((A+) = ABCH, (G+) = G.\)
  - \textit{YES.}\)

Uses of attribute set closures

- Determining \textit{superkeys and candidate keys}

- Determining if \( \alpha \rightarrow \beta \) is a valid FD
  - Check if \( \alpha^+ \) contains \( \beta \)

- Can be used to compute \( F^+ \)