Chapter 6: Relational Database Design

- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Decomposition Using Functional Dependencies
- Second Normal Form
- Third Normal Form and BC Normal Form
- Decomposition Using Multivalued Dependencies
- More Normal Form
- Database-Design Process
- Modeling Temporal Data
Students-course database

Student <U, F>

\[ U = \{ \text{Sno, Sdept, Mname, Cname, Grade} \} \]

\[ F = \{ \text{Sno} \to \text{Sdept}, \text{Sdept} \to \text{Mname}, \]
\[(\text{Sno, Cname}) \to \text{Grade} \} \]
Bad design with following Problems

- Data Redundancy
- Update Anomalies
- Insertion Anomalies
- Deletion Anomalies
Good design

- Goal of relational schema design is to avoid anomalies and redundancy.
- Eliminate data dependency
Schemas after decomposition

- \( S(\text{Sno, Sdept, Sno} \rightarrow \text{Sdept}) \)
- \( SC(\text{Sno, Cname, Grade, (Sno, Cname)} \rightarrow \text{Grade}) \)
- \( DEPT(\text{Sdept, Mname, Sdept} \rightarrow \text{Mname}) \)
The Banking Schema

- \textit{branch} = (\textit{branch\_name}, \textit{branch\_city}, \textit{assets})
- \textit{customer} = (\textit{customer\_id}, \textit{customer\_name}, \textit{customer\_street}, \textit{customer\_city})
- \textit{loan} = (\textit{loan\_number}, \textit{amount})
- \textit{account} = (\textit{account\_number}, \textit{balance})
- \textit{employee} = (\textit{employee\_id}, \textit{employee\_name}, \textit{telephone\_number}, \textit{start\_date})
- \textit{dependent\_name} = (\textit{employee\_id}, \textit{dname})
- \textit{account\_branch} = (\textit{account\_number}, \textit{branch\_name})
- \textit{loan\_branch} = (\textit{loan\_number}, \textit{branch\_name})
- \textit{borrower} = (\textit{customer\_id}, \textit{loan\_number})
- \textit{depositor} = (\textit{customer\_id}, \textit{account\_number})
- \textit{cust\_banker} = (\textit{customer\_id}, \textit{employee\_id}, \textit{type})
- \textit{works\_for} = (\textit{worker\_employee\_id}, \textit{manager\_employee\_id})
- \textit{payment} = (\textit{loan\_number}, \textit{payment\_number}, \textit{payment\_date}, \textit{payment\_amount})
- \textit{savings\_account} = (\textit{account\_number}, \textit{interest\_rate})
- \textit{checking\_account} = (\textit{account\_number}, \textit{overdraft\_amount})
Combine Schemas?

- Suppose we combine borrower and loan to get
  \[ \text{bor\_loan} = (\text{customer\_id}, \text{loan\_number}, \text{amount}) \]

- Result is possible repetition of information (L-100 in example below)
A Combined Schema Without Repetition

- Consider combining \textit{loan\_branch} and \textit{loan}

\[ \text{loan\_amt\_br} = (\text{loan\_number}, \text{amount}, \text{branch\_name}) \]

- No repetition (as suggested by example below)
What About Smaller Schemas?

- Suppose we had started with bor_loan. How would we know to split up (decompose) it into borrower and loan?
- Write a rule “if there were a schema (loan_number, amount), then loan_number would be a candidate key”
- Denote as a functional dependency:
  \[ \text{loan_number} \rightarrow \text{amount} \]
- In bor_loan, because loan_number is not a candidate key, the amount of a loan may have to be repeated. This indicates the need to decompose bor_loan.
- Not all decompositions are good. Suppose we decompose employee into
  \[ \text{employee1} = (employee\_id, employee\_name) \]
  \[ \text{employee2} = (employee\_name, telephone\_number, start\_date) \]
- The next slide shows how we lose information -- we cannot reconstruct the original employee relation -- and so, this is a lossy decomposition.
A Lossy Decomposition
First Normal Form

- Domain is atomic if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in first normal form if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account
  - We assume all relations are in first normal form
Atomicity is actually a property of how the elements of the domain are used.

- Example: Strings would normally be considered indivisible.
- Suppose that students are given roll numbers which are strings of the form CS0012 or EE1127.
- If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
- Doing so is a bad idea: leads to encoding of information in application program rather than in the database.
Goal — Devise a Theory for the Following

- Decide whether a particular relation \( R \) is in “good” form.
- In the case that a relation \( R \) is not in “good” form, decompose it into a set of relations \( \{R_1, R_2, ..., R_n\} \) such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies
Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a key.
Functional Dependencies (Cont.)

- Let $R$ be a relation schema
  \[ \alpha \subseteq R \text{ and } \beta \subseteq R \]
- The functional dependency
  \[ \alpha \rightarrow \beta \]
  holds on $R$ if and only if for any legal relations $r(R)$, whenever any two tuples $t_1$ and $t_2$ of $r$ agree on the attributes $\alpha$, they also agree on the attributes $\beta$. That is,
  \[ t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta] \]
- Example: Consider $r(A,B)$ with the following instance of $r$.

\[
\begin{array}{cc}
1 & 4 \\
1 & 5 \\
3 & 7 \\
\end{array}
\]

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.
Functional Dependencies (Cont.)

- $K$ 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 $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  \[
  \text{bor\_loan} = (\text{customer\_id, loan\_number, amount})
  \]

  We expect this functional dependency to hold:

  \[
  \text{loan\_number} \rightarrow \text{amount}
  \]

  but would not expect the following to hold:

  \[
  \text{amount} \rightarrow \text{customer\_name}
  \]
Use of Functional Dependencies

- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
    - If a relation \( r \) is legal under a set \( F \) of functional dependencies, we say that \( r \) satisfies \( F \).
  - specify constraints on the set of legal relations
    - We say that \( F \) holds on \( R \) if all legal relations on \( R \) satisfy the set of functional dependencies \( F \).

- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of \( loan \) may, by chance, satisfy \( amount \rightarrow customer\_name \).
A functional dependency is trivial if it is satisfied by all instances of a relation

Example:
- \( \text{customer}_\text{name}, \text{loan}_\text{number} \rightarrow \text{customer}_\text{name} \)
- \( \text{customer}_\text{name} \rightarrow \text{customer}_\text{name} \)

In general, \( \alpha \rightarrow \beta \) is trivial if \( \beta \subseteq \alpha \)
Second Normal Form

- Full dependency
  
  if $\alpha \rightarrow A$ and no subset $\beta$ of $\alpha$, $\beta \rightarrow A$, then $\alpha \_F \rightarrow A$

- Partial dependency

  if $\alpha \rightarrow A$, for some proper subset $\beta$ of $\alpha$, $\beta \rightarrow A$, then $\alpha \_P \rightarrow A$

- A relation schema $R$ is in 2NF if no non-prime attribute $A$ is partially dependent on any candidate key for $R$. 
Closure of a Set of Functional Dependencies

- Given a set $F$ of functional dependencies, there are certain other functional dependencies that are logically implied by $F$.
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of all functional dependencies logically implied by $F$ is the closure of $F$.
- We denote the closure of $F$ by $F^+$.
- $F^+$ is a superset of $F$. 
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., $\beta \subseteq \alpha$)
- $\alpha$ is a superkey for $R$

Example schema not in BCNF:

$borrowed_loan = (\text{customer_id, loan_number, amount})$

because $\text{loan_number} \rightarrow \text{amount}$ holds on $bore_loan$ but $\text{loan_number}$ is not a superkey
Decomposing a Schema into BCNF

- Suppose we have a schema $R$ and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

We decompose $R$ into:
- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

- In our example,
  - $\alpha = \text{loan_number}$
  - $\beta = \text{amount}$

and $\text{bor_loan}$ is replaced by
- $(\alpha \cup \beta) = (\text{loan_number, amount})$
- $(R - (\beta - \alpha)) = (\text{customer_id, loan_number})$
BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation.
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that all functional dependencies hold, then that decomposition is dependency preserving.
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as third normal form.
Third Normal Form: Motivation

- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems; we will see examples later)
  - But functional dependencies can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.
Third Normal Form

- 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., $\beta \in \alpha$)
  - $\alpha$ is a superkey for $R$
  - Each attribute $A$ in $\beta - \alpha$ is contained in a candidate key for $R$.
    (NOTE: each attribute may be in a different candidate key)
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation.
3NF Example

- Relation R:
  - $R = (J, K, L)$
  - $F = \{JK \rightarrow L, L \rightarrow K\}$
- Two candidate keys: JK and JL
- R is in 3NF
  - $JK \rightarrow L$ \hspace{1em} JK is a superkey
  - $L \rightarrow K$ \hspace{1em} K is contained in a candidate key
Redundancy in 3NF

- There is some redundancy in this schema
- Example of problems due to redundancy in 3NF
  - $R = (J, K, L)$
  - $F = \{JK \rightarrow L, L \rightarrow K\}$

<table>
<thead>
<tr>
<th></th>
<th>J</th>
<th>L</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j_1$</td>
<td>$l_1$</td>
<td>$k_1$</td>
<td></td>
</tr>
<tr>
<td>$j_2$</td>
<td>$l_1$</td>
<td>$k_1$</td>
<td></td>
</tr>
<tr>
<td>$j_3$</td>
<td>$l_1$</td>
<td>$k_1$</td>
<td></td>
</tr>
<tr>
<td>null</td>
<td>$l_2$</td>
<td>$k_2$</td>
<td></td>
</tr>
</tbody>
</table>

- Repetition of information (e.g., the relationship $l_1, k_1$)
- Need to use null values (e.g., to represent the relationship $l_2, k_2$ where there is no corresponding value for J).
Goals of Normalization

- Let \( R \) be a relation scheme with a set \( F \) of functional dependencies.
- Decide whether a relation scheme \( R \) is in “good” form.
- In the case that a relation scheme \( R \) is not in “good” form, decompose it into a set of relation schemes \( \{R_1, R_2, \ldots, R_n\} \) such that
  - each relation scheme is in good form
  - the decomposition is a lossless-join decomposition
  - Preferably, the decomposition should be dependency preserving.
How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized.
- Consider a database

  \[ \text{classes (course, teacher, book)} \]

  such that \((c, t, b) \in \text{classes}\) means that \(t\) is qualified to teach \(c\), and \(b\) is a required textbook for \(c\).

- The database is supposed to list for each course the set of teachers any one of which can be the course’s instructor, and the set of books, all of which are required for the course (no matter who teaches it).
How good is BCNF? (Cont.)

<table>
<thead>
<tr>
<th>course</th>
<th>teacher</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>Avi</td>
<td>DB Concepts</td>
</tr>
<tr>
<td>database</td>
<td>Avi</td>
<td>Ullman</td>
</tr>
<tr>
<td>database</td>
<td>Hank</td>
<td>DB Concepts</td>
</tr>
<tr>
<td>database</td>
<td>Hank</td>
<td>Ullman</td>
</tr>
<tr>
<td>database</td>
<td>Sudarshan</td>
<td>DB Concepts</td>
</tr>
<tr>
<td>database</td>
<td>Sudarshan</td>
<td>Ullman</td>
</tr>
<tr>
<td>operating systems</td>
<td>Avi</td>
<td>OS Concepts</td>
</tr>
<tr>
<td>operating systems</td>
<td>Avi</td>
<td>Stallings</td>
</tr>
<tr>
<td>operating systems</td>
<td>Pete</td>
<td>OS Concepts</td>
</tr>
<tr>
<td>operating systems</td>
<td>Pete</td>
<td>Stallings</td>
</tr>
</tbody>
</table>

classes

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if Marilyn is a new teacher that can teach database, two tuples need to be inserted
  (database, Marilyn, DB Concepts)
  (database, Marilyn, Ullman)
Therefore, it is better to decompose classes into:

```
course         teacher
---------------
database      Avi
  database    Hank
  database    Sudarshan
operating systems  Avi
    operating systems  Jim
```

*teaches*

```
course     book
------------
database   DB Concepts
  database  Ullman
  operating systems  OS Concepts
    operating systems  Shaw
```

*text*

This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later.
Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved

- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.
Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.
  Can specify FDs using assertions, but they are expensive to test
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key.
Multivalued Dependencies (MVDs)

- Let $R$ be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$.

The multivalued dependency

$$\alpha \rightarrow \beta$$

holds on $R$ if in any legal relation $r(R)$, for all pairs for tuples $t_1$ and $t_2$ in $r$ such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples $t_3$ and $t_4$ in $r$ such that:

- $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
- $t_3[\beta] = t_1[\beta]$
- $t_3[R - \beta] = t_2[R - \beta]$
- $t_4[\beta] = t_2[\beta]$
- $t_4[R - \beta] = t_1[R - \beta]$
MVD (Cont.)

• Tabular representation of $\alpha \rightarrow \rightarrow \beta$

<table>
<thead>
<tr>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1 \ldots a_i$</td>
<td>$a_1 \ldots a_i$</td>
<td>$a_1 \ldots a_i$</td>
<td>$a_1 \ldots a_i$</td>
</tr>
<tr>
<td>$a_i + 1 \ldots a_j$</td>
<td>$b_i + 1 \ldots b_j$</td>
<td>$a_i + 1 \ldots a_j$</td>
<td>$b_i + 1 \ldots b_j$</td>
</tr>
<tr>
<td>$a_j + 1 \ldots a_n$</td>
<td>$b_j + 1 \ldots b_n$</td>
<td>$b_j + 1 \ldots b_n$</td>
<td>$a_j + 1 \ldots a_n$</td>
</tr>
</tbody>
</table>
Example

- Let $R$ be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.
  
  $Y, Z, W$

- We say that $Y \rightarrow\rightarrow Z$ ($Y$ multidetermines $Z$) if and only if for all possible relations $r (R)$
  
  $\langle y_1, z_1, w_1 \rangle \in r$ and $\langle y_1, z_2, w_2 \rangle \in r$

  then

  $\langle y_1, z_1, w_2 \rangle \in r$ and $\langle y_1, z_2, w_1 \rangle \in r$

- Note that since the behavior of $Z$ and $W$ are identical it follows that

  $Y \rightarrow\rightarrow Z$ if $Y \rightarrow\rightarrow W$
Example (Cont.)

- In our example:
  
  \[
  \text{course} \rightarrow\rightarrow \text{teacher} \\
  \text{course} \rightarrow\rightarrow \text{book}
  \]

- The above formal definition is supposed to formalize the notion that given a particular value of \( Y (\text{course}) \) it has associated with it a set of values of \( Z (\text{teacher}) \) and a set of values of \( W (\text{book}) \), and these two sets are in some sense independent of each other.

- Note:
  
  - If \( Y \rightarrow Z \) then \( Y \rightarrow\rightarrow Z \)
  
  - Indeed we have (in above notation) \( Z_1 = Z_2 \)
    The claim follows.
Use of Multivalued Dependencies

- We use multivalued dependencies in two ways:
  1. To test relations to determine whether they are legal under a given set of functional and multivalued dependencies.
  2. To specify constraints on the set of legal relations. We shall thus concern ourselves only with relations that satisfy a given set of functional and multivalued dependencies.
- If a relation $r$ fails to satisfy a given multivalued dependency, we can construct a relations $r'$ that does satisfy the multivalued dependency by adding tuples to $r$. 
Theory of MVDs

- From the definition of multivalued dependency, we can derive the following rule:
  - If $\alpha \rightarrow \beta$, then $\alpha \rightarrow\!\!\!\!\!\!\!\!\!\rightarrow \beta$

That is, every functional dependency is also a multivalued dependency

- The **closure** $D^+$ of $D$ is the set of all functional and multivalued dependencies logically implied by $D$.
  - We can compute $D^+$ from $D$, using the formal definitions of functional dependencies and multivalued dependencies.
  - We can manage with such reasoning for very simple multivalued dependencies, which seem to be most common in practice.
  - For complex dependencies, it is better to reason about sets of dependencies using a system of inference rules (see Appendix C).
Fourth Normal Form

- A relation schema $R$ is in 4NF with respect to a set $D$ of functional and multivalued dependencies if for all multivalued dependencies in $D^+$ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
  - $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  - $\alpha$ is a superkey for schema $R$
  - If a relation is in 4NF it is in BCNF
Restriction of Multivalued Dependencies

- The restriction of $D$ to $R_i$ is the set $D_i$ consisting of
  - All functional dependencies in $D^+$ that include only attributes of $R_i$
  - All multivalued dependencies of the form
    \[ \alpha \rightarrow (\beta \cap R_i) \]
    where $\alpha \subseteq R_i$ and $\alpha \rightarrow \beta$ is in $D^+$
Further Normal Forms

- **Join dependencies** generalize multivalued dependencies
  - lead to **project-join normal form** (PJNF) (also called **fifth normal form**)
- A class of even more general constraints, leads to a normal form called **domain-key normal form**.
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
- Hence rarely used

\[ 1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF \]
Overall Database Design Process

- We have assumed schema $R$ is given
  - $R$ could have been generated when converting E-R diagram to a set of tables.
  - $R$ could have been a single relation containing all attributes that are of interest (called universal relation).
  - Normalization breaks $R$ into smaller relations.
  - $R$ could have been the result of some ad hoc design of relations, which we then test/convert to normal form.
ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.
- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example: an employee entity with attributes department_number and department_address, and a functional dependency department_number → department_address
  - Good design would have made department an entity
- Functional dependencies from non-key attributes of a relationship set possible, but rare --- most relationships are binary
Denormalization for Performance

- May want to use non-normalized schema for performance
- For example, displaying `customer_name` along with `account_number` and `balance` requires join of `account` with `depositor`
- Alternative 1: Use denormalized relation containing attributes of `account` as well as `depositor` with all above attributes
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: use a materialized view defined as `account \frown depositor`
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors
Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:

  Instead of `earnings (company_id, year, amount )`, use

  - `earnings_2004, earnings_2005, earnings_2006, etc.`, all on the schema
    `(company_id, earnings)`.
    - Above are in BCNF, but make querying across years difficult and needs new table each year
    - Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
    - Is an example of a **crosstab**, where values for one attribute become column names
    - Used in spreadsheets, and in data analysis tools
Modeling Temporal Data

- *Temporal data* have an association time interval during which the data are *valid*.

- A *snapshot* is the value of the data at a particular point in time.

- Several proposals to extend ER model by adding valid time to:
  - attributes, e.g. address of a customer at different points in time
  - entities, e.g. time duration when an account exists
  - relationships, e.g. time during which a customer owned an account

- But no accepted standard

- Adding a temporal component results in functional dependencies like

  \[
  \text{customer}_id \rightarrow \text{customer}_street, \text{customer}_city
  \]
  
  not to hold, because the address varies over time

- A *temporal functional dependency* \( X \rightarrow Y \) holds on schema \( R \) if the functional dependency \( X \rightarrow Y \) holds on all snapshots for all legal instances \( r (R) \)
Modeling Temporal Data (Cont.)

- In practice, database designers may add start and end time attributes to relations
  - E.g. \textit{course}(course\_id, course\_title) $\rightarrow$
    
    \textit{course}(course\_id, course\_title, start, end)
  
  - Constraint: no two tuples can have overlapping valid times
    
    - Hard to enforce efficiently

- Foreign key references may be to current version of data, or to data at a point in time
  - E.g. student transcript should refer to course information at the time the course was taken
Figure 7.6
Figure 7.7