CMSC 461, Database Management Systems
Spring 2018

Lecture 13 - Chapter 8 Relational Database Design Part 3

These slides are based on “Database System Concepts” 6th edition book and are a modified version of the slides which accompany the book (http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html), in addition to the 2009/2012 CMSC 461 slides by Dr. Kalpakis
Logistics

- Midterm 3/14/2018
Lecture Outline

• *Functional Dependencies Review*
• BCNF Decomposition
• Database Design Process
• Midterm Review
Functional Dependencies

We represent constraints by using keys
- super keys
- primary keys
- candidate keys

We can also represent constraints using functional dependencies
Functional Dependencies

- A *functional dependency* is a relationship between two attributes, where $\beta$ is functionally dependent on $\alpha$
- $\alpha$ is usually the primary key
- For every valid instance of $\alpha$, that value uniquely determines the value of $\beta$. 

\[ \alpha \rightarrow \beta \]
Functional Dependencies

What are the dependencies in Table Foo?

| Table Foo: |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| a1 | b1 | c1 | d1 | e1 |
| a2 | b1 | c2 | d2 | e1 |
| a3 | b2 | c1 | d1 | e1 |
| a4 | b2 | c2 | d2 | e1 |
Functional Dependencies

Can we say this:

\[ A \rightarrow B, \quad A \rightarrow C, \quad A \rightarrow D, \quad A \rightarrow E \]

<table>
<thead>
<tr>
<th>Table Foo:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>a1</td>
</tr>
<tr>
<td>a2</td>
</tr>
<tr>
<td>a3</td>
</tr>
<tr>
<td>a4</td>
</tr>
</tbody>
</table>
Functional Dependencies

*If we can say this:*

\[ A \rightarrow B, \quad A \rightarrow C, \quad A \rightarrow D, \quad A \rightarrow E \]

*Then we can say:*

\[ A \rightarrow BC \quad \text{(or any other subset of ABCDE)} \]

---

**Table Foo:**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a2</td>
<td>a2</td>
<td>b1</td>
<td>c2</td>
<td>d2</td>
<td>e1</td>
</tr>
<tr>
<td>a3</td>
<td>a3</td>
<td>b2</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a4</td>
<td>a4</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
<td>e1</td>
</tr>
</tbody>
</table>

Based on and image from "Database System Concepts" book and slides, 6th edition, example table from https://opentextbc.ca/dbdesign01/chapter/chapter-11-functional-dependencies/
Functional Dependencies

We can summarize this as

\[ A \rightarrow BCDE \]

Table Foo:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c2</td>
<td>d2</td>
<td>e1</td>
</tr>
<tr>
<td>a3</td>
<td>b2</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a4</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
<td>e1</td>
</tr>
</tbody>
</table>
Functional Dependencies

Other dependencies we can observe?

Table Foo:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c2</td>
<td>d2</td>
<td>e1</td>
</tr>
<tr>
<td>a3</td>
<td>b2</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a4</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
<td>e1</td>
</tr>
</tbody>
</table>
Functional Dependency Theory

We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
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:
    Given a schema $r(A,B,C)$
    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 $F$.
Closure of a Set of Functional Dependencies

- We can find $F^+$, the closure of $F$, by repeatedly applying
  **Armstrong’s Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ \hspace{1cm} (reflexivity)
  - if $\alpha \rightarrow \beta$, then $\gamma \alpha \rightarrow \gamma \beta$ \hspace{1cm} (augmentation)
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ \hspace{1cm} (transitivity)
- These rules are
  - **sound** (generate only functional dependencies that actually hold), and
  - **complete** (generate all functional dependencies that hold).
Computing F+

• To compute the closure of a set of functional dependencies F:

\[ F^+ = F \]

repeat
for each functional dependency \( f \) in \( F^+ \)
   apply reflexivity and augmentation rules on \( f \)
   add the resulting functional dependencies to \( F^+ \)
for each pair of functional dependencies \( f_1 \) and \( f_2 \) in \( F^+ \)
   if \( f_1 \) and \( f_2 \) can be combined using transitivity
      then add the resulting functional dependency to \( F^+ \)
until \( F^+ \) does not change any further

Based on and image from "Database System Concepts" book and slides, 6th edition
Closure of a set of Functional Dependencies

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

The above rules can be inferred from Armstrong’s axioms.
Closure of a set of Functional Dependencies Example

- \( R = (A, B, C, G, H, I) \)
- \( F = \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \} \)

- 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 union rule, since \( CG \rightarrow H \) and \( CG \rightarrow I \), implies \( CG \rightarrow HI \)
Closure of Attribute Sets

• Given a set of attributes $\alpha$, define the closure of $\alpha$ under $F$ (denoted by $\alpha^+$) as the set of attributes that are functionally determined by $\alpha$ under $F$.

• Algorithm to compute $\alpha^+$, the closure of $\alpha$ under $F$

\[
\text{result} := \alpha;
\text{while (changes to result) do}
\text{for each } \beta \rightarrow \gamma \text{ in } F \text{ do}
\text{begin}
\text{if } \beta \subseteq \text{result} \text{ then } \text{result} := \text{result} \cup \gamma
\text{end}
\]
Closure of Attribute Sets Example

- \( R = (A, B, C, G, H, I) \)
- \( F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\} \)
- \((AG)^+\)
  1. \(\text{result} = AG\)
  2. \(\text{result} = ABCG \ (A \rightarrow C \text{ and } A \rightarrow B)\)
  3. \(\text{result} = ABCGH \ (CG \rightarrow H \text{ and } CG \subseteq AGBC)\)
  4. \(\text{result} = ABCGHI \ (CG \rightarrow I \text{ and } CG \subseteq AGBCH)\)
Closure of Attribute Sets Uses

There are several uses of the attribute closure algorithm:

- **Testing for superkey:**
  - To test if $\alpha$ is a superkey, we compute $\alpha^+$, and check if $\alpha^+$ contains all attributes of $R$.

- **Testing functional dependencies**
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in $F^+$), just check if $\beta \subseteq \alpha^+$.
  - That is, we compute $\alpha^+$ by using attribute closure, and then check if it contains $\beta$.
  - Is a simple and cheap test, and very useful

- **Computing closure of $F$**
  - For each $\gamma \subseteq R$, we find the closure $\gamma^+$, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$. 

Based on and image from "Database System Concepts" book and slides, 6th edition
Closure of Attribute Sets Examples

- $R = (\text{name}, \text{color}, \text{category}, \text{department}, \text{price})$
- $F = \{\text{name} \rightarrow \text{color} \\\ \text{category} \rightarrow \text{department} \\\ \text{color}, \text{category} \rightarrow \text{price}\}$

You find:
- $\text{name}^+$
- $\{\text{name}, \text{category}\}^+$
- $\{\text{color}\}^+$
Lecture Outline

• Functional Dependencies Review
• $BCNF$ Decomposition
• Database Design Process
• Midterm Review
Lossless-join Decomposition

- For the case of $R = (R_1, R_2)$, we require that for all possible relations $r$ on schema $R$,
  $$r = \prod_{R_1} (r) \prod_{R_2} (r)$$
- A decomposition of $R$ into $R_1$ and $R_2$ is lossless join if at least one of the following dependencies is in $F^+$:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies.

Example

- \( R = (A, B, C) \)
  \( F = \{ A \rightarrow B, B \rightarrow C \} \)
  - Can be decomposed in two different ways

- \( R_1 = (A, B), \quad R_2 = (B, C) \)
  - Lossless-join decomposition:
    \( R_1 \cap R_2 = \{ B \} \) and \( B \rightarrow BC \)
  - Dependency preserving

- \( R_1 = (A, B), \quad R_2 = (A, C) \)
  - Lossless-join decomposition:
    \( R_1 \cap R_2 = \{ A \} \) and \( A \rightarrow AB \)
  - Not dependency preserving
    (cannot check \( B \rightarrow C \) without computing \( R_1 \Join R_2 \))
Let $F_i$ be the set of dependencies $F^+$ that include only attributes in $R_i$.

- A decomposition is **dependency preserving**, if

\[
(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+
\]

- If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.

Based on “Database System Concepts” book and slides, 6th edition
Testing for Dependency Preservation

To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of $R$ into $R_1, R_2, \ldots, R_n$ we apply the following test (with attribute closure done with respect to $F$)

1. $result = \alpha$
2. while (changes to $result$) do
   for each $R_i$ in the decomposition
   
   $t = (result \cap R_i)^+ \cap R_i$
   
   $result = result \cup t$

3. If $result$ contains all attributes in $\beta$, then the functional dependency $\alpha \rightarrow \beta$ is preserved.
Testing for Dependency Preservation

- We apply the test on all dependencies in $F$ to check if a decomposition is dependency preserving.
- This procedure takes polynomial time, instead of the exponential time required to compute $F^+$ and $((F_1 \cup F_2 \cup \ldots \cup F_n)^+)$.

Example

- $R = (A, B, C )$
  $F = \{A \rightarrow B$
  $B \rightarrow C\}$
  Key = \{A\}
- $R$ is not in BCNF
- Decomposition $R_1 = (A, B), \quad R_2 = (B, C)$
  - $R_1$ and $R_2$ in BCNF
  - Lossless-join decomposition
  - Dependency preserving
Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  1. compute $\alpha^+$ (the attribute closure of $\alpha$), and
  2. verify that it includes all attributes of $R$, that is, it is a superkey of $R$. 

Based on “Database System Concepts” book and slides, 6th edition
Testing for BCNF

- **Simplified test**: To check if a relation schema $R$ is in BCNF, it suffices to check only the dependencies in the given set $F$ for violation of BCNF, rather than checking all dependencies in $F^+$.
  - If none of the dependencies in $F$ causes a violation of BCNF, then none of the dependencies in $F^+$ will cause a violation of BCNF either.
Testing for BCNF

- However, simplified test using only $F$ is incorrect when testing a relation in a decomposition of $R$
  - Consider $R = (A, B, C, D, E)$, with $F = \{ A \rightarrow B, BC \rightarrow D \}$
    - Decompose $R$ into $R_1 = (A,B)$ and $R_2 = (A,C,D, E)$
    - Neither of the dependencies in $F$ contain only attributes from $(A, C, D, E)$ so we might be misled into thinking $R_2$ satisfies BCNF.
    - In fact, dependency $AC \rightarrow D$ in $F^+$ shows $R_2$ is not in BCNF.

Based on “Database System Concepts” book and slides, 6th edition
Testing Decomposition for BCNF

- To check if a relation $R_i$ in a decomposition of $R$ is in BCNF,
  - Either test $R_i$ for BCNF with respect to the restriction of $F$ to $R_i$ (that is, all FDs in $F^+$ that contain only attributes from $R_i$)
  - or use the original set of dependencies $F$ that hold on $R$, but with the following test:
    - for every set of attributes $\alpha \subseteq R_i$, check that $\alpha^+$ (the attribute closure of $\alpha$) either includes no attribute of $R_i - \alpha$, or includes all attributes of $R_i$.
  - If the condition is violated by some $\alpha \rightarrow \beta$ in $F$, the dependency
    $$\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$$
    can be shown to hold on $R_i$, and $R_i$ violates BCNF.
- We use above dependency to decompose $R_i$
Example of BCNF Decomposition

- $R = (A, B, C)$
  
  $F = \{ A \rightarrow B, \ B \rightarrow C \}$
  
  Key = $\{A\}$

- $R$ is not in BCNF ($B \rightarrow C$ but $B$ is not superkey)

- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A, B)$
Example of BCNF Decomposition

- **class** (course_id, title, dept_name, credits, sec_id, semester, year, building, room_number, capacity, time_slot_id)

- Functional dependencies:
  - course_id → title, dept_name, credits
  - building, room_number → capacity
  - course_id, sec_id, semester, year → building, room_number, time_slot_id

Based on “Database System Concepts” book and slides, 6th edition
Example of BCNF Decomposition

- A candidate key \{\textit{course\_id}, \textit{sec\_id}, \textit{semester}, \textit{year}\}.

- BCNF Decomposition:
  - \textit{course\_id} → \textit{title}, \textit{dept\_name}, \textit{credits} holds
    - but \textit{course\_id} is not a superkey.
  - We replace \textit{class} by:
    - \textit{course}(\textit{course\_id}, \textit{title}, \textit{dept\_name}, \textit{credits})
    - \textit{class-1}(\textit{course\_id}, \textit{sec\_id}, \textit{semester}, \textit{year}, \textit{building}, \textit{room\_number}, \textit{capacity}, \textit{time\_slot\_id})
BCNF Decomposition

- course is in BCNF
  - How do we know this?
- building, room_number → capacity holds on class-1
  - but \{building, room_number\} is not a superkey for class-1.
  - We replace class-1 by:
    - classroom (building, room_number, capacity)
    - section (course_id, sec_id, semester, year, building, room_number, time_slot_id)
- classroom and section are in BCNF.
BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- \( R = (J, K, L) \)
  \( F = \{ JK \rightarrow L, \ L \rightarrow K \} \)
  Two candidate keys = JK and JL
- \( R \) is not in BCNF
- Any decomposition of \( R \) will fail to preserve
  \( JK \rightarrow L \)
  This implies that testing for \( JK \rightarrow L \) requires a join

Based on “Database System Concepts” book and slides, 6th edition
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

Based on "Database System Concepts" book and slides, 6th edition
Design Goals

- 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, (and currently not supported by any of the widely used databases!)
- 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.

Based on “Database System Concepts” book and slides, 6th edition
Lecture Outline

• Functional Dependencies Review
• BCNF Decomposition
• *Database Design Process*
• Midterm Review
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.
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_name` and `building`, and a functional dependency `department_name → building`
- 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 `prereqs` along with `course_id`, and `title` requires join of `course` with `prereq`
- Alternative 1: Use denormalized relation containing attributes of `course` as well as `prereq` 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
  - `course` `prereq`
  - 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.
Lecture Outline

- Functional Dependencies Review
- BCNF Decomposition
- 3NF Decomposition
- Multivalued Decomposition
- Fourth Normal Form
- Database Design Process
- *Midterm Review*
Midterm Review

Joins

SQL JOINS

SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key

SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key

SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key

SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL

SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL

SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL

SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL

© C.L. Moffatt, 2008
Joined Relations

- Join operations take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join.
- The join operations are typically used as subquery expressions in the from clause.

Based on “Database System Concepts” book and slides, 6th edition
Joined Relations

Cartesian with where clause

Select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID;

Based on “Database System Concepts” book and slides, 6th edition
Joined Relations

Natural Join

Select name, course_id
from instructor natural join teaches;
Joined Relations

- There is also `join` with `using` clause
  
  ```sql
  Select name, course_id from instructor join teaches using (ID);
  ```

- You must specify list of attributes to join upon

- Both relations must have the same name

- Similar to natural join except:
  - Not all attributes that are the same are joined upon
Joined Relations

- There is also `join` with `on` condition
  - `Select name, course_id`
  - `from instructor join teaches on (instructor.ID = teaches.ID)`

- Arbitrary join condition

- Similar to using `where` clause to specify join condition
  - The `on` condition behaves differently for outer joins
Join Example

```sql
select * from course, prereq where course.course_id = prereq.course_id;
```

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>course_id</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-301</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-190</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

2 rows in set (0.00 sec)
## Join Example

```sql
select * from course natural join prereq;
```

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

2 rows in set (0.00 sec)
Join Example

```sql
select * from course join prereq
using(course_id);
```

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

2 rows in set (0.00 sec)
### Join Example

```sql
select * from course join prereq on course.course_id = prereq.course_id;
```

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>course_id</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-301</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-190</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

2 rows in set (0.01 sec)
Outer Joins

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses null values.
- **inner join** – join operations that do not preserve non-matched tuples
Left Outer Join

```sql
select * from course natural left outer join prereq;
```

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
<td>NULL</td>
</tr>
</tbody>
</table>

3 rows in set (0.00 sec)
Right Outer Join

```
select * from course natural right outer join prereq;
```

<table>
<thead>
<tr>
<th>course_id</th>
<th>prereq_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>BIO-101</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
</tr>
<tr>
<td>CS-190</td>
<td>CS-101</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
</tr>
<tr>
<td>CS-347</td>
<td>CS-101</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

3 rows in set (0.00 sec)
Full Outer Join

```
select * from course natural full outer join prereq;
```

+-------------------+-----------------+--------------+--------+-----------------+
| course_id | title       | dept_name   | credits | prereq_id       |
+-----------+-------------+-------------+---------+-----------------+
| BIO-301   | Genetics    | Biology     | 4       | BIO-101         |
| CS-190    | Game Design | Comp. Sci.  | 4       | CS-101          |
| CS-315    | Robotics    | Comp. Sci.  | 3       | NULL            |
| BIO-301   | BIO-101     | Genetics    | Biology | 4               |
| CS-190    | CS-101      | Game Design | Comp. Sci. | 4            |
| CS-347    | CS-101      | NULL        | NULL    | NULL            |
+-----------+-------------+-------------+---------+-----------------+

6 rows in set (0.00 sec)
A large organization has several parking lots which are used by staff. Each parking lot has a unique name, location, capacity, and number of floors. Each parking lot has parking spaces which are uniquely identified within that lot using a space number. Each space is located on a particular floor within the lot. Members of staff are assigned the use of a parking space. Each member of staff has a unique id number, name, telephone, and vehicle license plate number.
Midterm Review

How should we begin?
Midterm Review

How should we begin?
Create an ER Diagram

What are the entity sets?
What are the relation sets?
Attributes?
Midterm Review
Midterm Review

Reduce to a relational model. Underline primary keys and use a + to indicate foreign keys.
Midterm Review

Staff \( (staff\_no, \text{first\_name, last\_name, phone, license\_plate}) \)
Assignment\( (staff\_no(+), lot\_name(+), \text{space\_number}+) \)
Space \( (lot\_name(+), space\_number, floor\_number) \)
Lot \( (lot\_name, \text{location, capacity, floors}) \)
Midterm Review

List the set of functional dependencies for each relation in your model that originated from the problem statement above. For each functional dependency, use the form $\alpha \rightarrow \beta$ where $\alpha$ and $\beta$ are one or more attributes from your relation schema.

Indicate the relation and the set of functional dependencies using this pattern:
Relation name: F = \{a→b, c→d \}
Midterm Review

Staff:
\[ F = \{ \text{staff\_no} \rightarrow \text{first\_name}, \text{last\_name}, \text{phone}, \text{license\_plate} \} \]

Assignment:
\[ F = \{ \text{staff\_no}, \text{lot\_name} \rightarrow \text{space\_number} \} \]

Space:
\[ F = \{ \text{lot\_name}, \text{space\_number} \rightarrow \text{floor\_number} \} \]

Lot:
\[ F = \{ \text{lot\_name} \rightarrow \text{location}, \text{capacity}, \text{floors} \} \]
Midterm Review

Do we pass the test for BCNF?

Why or why not?
Midterm Review

Do we pass the test for BCNF?

Yes, because they are all non-trivial and all superkeys.
Midterm Review

Based on the functional dependencies from above, use Armstrong’s Axioms and their derivatives, to derive three new functional dependencies.
Midterm Review

DECOMPOSITION

\( \text{lot\_name} \rightarrow \text{location, capacity, floors} \)
\( \text{lot\_name} \rightarrow \text{location, capacity} \)
\( \text{lot\_name} \rightarrow \text{floors} \)

REFLEXIVITY

\( \text{staff\_no} \rightarrow \text{staff\_no} \)

AUGMENTATION

\( \text{staff\_no} \rightarrow \text{license\_plate} \)
\( \text{staff\_no, phone} \rightarrow \text{license\_plate, phone} \)