Functional Dependencies
DATABASE DESIGN

How do we design a “good” database schema?

We want to ensure the integrity of the data.

We also want to get good performance.
### EXAMPLE DATABASE

The database represents student records in a tabular format:

```sql
student(sid, cid, room, grade, name, address)
```

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>room</th>
<th>grade</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-445</td>
<td>GHC 6115</td>
<td>A</td>
<td>Andy</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>456</td>
<td>15-721</td>
<td>GHC 8102</td>
<td>B</td>
<td>Tupac</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>789</td>
<td>15-445</td>
<td>GHC 6115</td>
<td>A</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
<tr>
<td>012</td>
<td>15-445</td>
<td>GHC 6115</td>
<td>C</td>
<td>Waka Flocka</td>
<td>Atlanta</td>
</tr>
<tr>
<td>789</td>
<td>15-721</td>
<td>GHC 8102</td>
<td>A</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
</tbody>
</table>
### EXAMPLE DATABASE

The database is represented as a table with the following columns:

- sid: Student ID
- cid: Course ID
- room: Room number
- grade: Grade
- name: Name
- address: Address

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>room</th>
<th>grade</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-445</td>
<td>GHC 6115</td>
<td>A</td>
<td>Andy</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>456</td>
<td>15-721</td>
<td>GHC 8102</td>
<td>B</td>
<td>Tupac</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>789</td>
<td>15-445</td>
<td>GHC 6115</td>
<td>A</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
<tr>
<td>012</td>
<td>15-445</td>
<td>GHC 6115</td>
<td>C</td>
<td>Waka Flocka</td>
<td>Atlanta</td>
</tr>
<tr>
<td>789</td>
<td>15-721</td>
<td>GHC 8102</td>
<td>A</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
</tbody>
</table>
REDUNDANCY PROBLEMS

Update Anomalies
→ If the room number changes, we need to make sure that we change all students records.

Insert Anomalies
→ May not be possible to add a student unless they’re enrolled in a course.

Delete Anomalies
→ If all the students enrolled in a course are deleted, then we lose the room number.
### Example Database

**student(sid,name,address)**

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Andy</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>456</td>
<td>Tupac</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>789</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
<tr>
<td>012</td>
<td>Waka Flocka</td>
<td>Atlanta</td>
</tr>
</tbody>
</table>

**courses(sid,cid,grade)**

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-415</td>
<td>A</td>
</tr>
<tr>
<td>456</td>
<td>15-721</td>
<td>B</td>
</tr>
<tr>
<td>789</td>
<td>15-415</td>
<td>A</td>
</tr>
<tr>
<td>012</td>
<td>15-415</td>
<td>C</td>
</tr>
<tr>
<td>789</td>
<td>15-721</td>
<td>A</td>
</tr>
</tbody>
</table>

**rooms(cid,room)**

<table>
<thead>
<tr>
<th>cid</th>
<th>room</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-415</td>
<td>GHC 6115</td>
</tr>
<tr>
<td>15-721</td>
<td>GHC 8102</td>
</tr>
</tbody>
</table>

---

**Why this decomposition is better and how to find it.**
TODAY'S AGENDA

Functional Dependencies
Canonical Cover
Schema Decomposition
FUNCTIONAL DEPENDENCIES

A functional dependency (FD) is a form of a constraint. Part of a relation's schema to define a valid instance.

Definition: $X \rightarrow Y$
→ The value of $X$ functionally defines the value of $Y$. 
FUNCTIONAL DEPENDENCIES

Formal Definition:
\[ X \rightarrow Y \Rightarrow (t_1[x] = t_2[x] \Rightarrow t_1[y] = t_2[y]) \]

If two tuples \((t_1, t_2)\) agree on the \(X\) attribute, then they must agree on the \(Y\) attribute too.

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Andy</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>456</td>
<td>Tupac</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>789</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
<tr>
<td>012</td>
<td>Waka Flocka</td>
<td>Atlanta</td>
</tr>
</tbody>
</table>
FUNCTIONAL DEPENDENCIES

Formal Definition:
→ X⇒Y ⇒ (t₁[x]=t₂[x] ⇒ t₁[y]=t₂[y])

If two tuples (t₁, t₂) agree on the X attribute, then they must agree on the Y attribute too.

\[ R₁(\text{sid}, \text{name}, \text{address}) \]

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Andy</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>456</td>
<td>Tupac</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>789</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
<tr>
<td>012</td>
<td>Waka Flocka</td>
<td>Atlanta</td>
</tr>
</tbody>
</table>

\[ X \quad Y \]

✔ sid⇒name
FUNCTIONAL DEPENDENCIES

FD is a constraint that allows instances for which the FD holds.

You can check if an FD is violated by an instance, but you cannot prove that an FD is part of the schema using an instance.

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Andy</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>456</td>
<td>Tupac</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>789</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
<tr>
<td>012</td>
<td>Waka Flocka</td>
<td>Atlanta</td>
</tr>
<tr>
<td>555</td>
<td>Andy</td>
<td>Providence</td>
</tr>
</tbody>
</table>
FUNCTIONAL DEPENDENCIES

Two FDs $X \rightarrow Y$ and $X \rightarrow Z$ can be written in shorthand as $X \rightarrowYZ$.

But $XY \rightarrow Z$ is not the same as the two FDs $X \rightarrow Z$ and $Y \rightarrow Z$. 
DEFINING FDS IN SQL (EX.1)

Make sure that no two students ever have the same id without the same name.

```
CREATE ASSERTION student-name 
CHECK (NOT EXISTS 
(SELECT * FROM students AS s1, 
students AS s2 
WHERE s1.sid = s2.sid 
AND s1.name <> s2.name))
```

FD₁: sid → name
DEFINING FDS IN SQL (EX.2)

Make sure that no two students ever have the same id without the same name and address.

CREATE ASSERTION student-name-addr SQL-92
CHECK (NOT EXISTS
(SELECT * FROM students AS s1,
     students AS s2
WHERE s1.sid = s2.sid
AND ((s1.name <> s2.name
     OR (s1.address <> s2.address)))))

FD₁: sid → name
FD₂: sid → address
SQL ASSERTIONS

As of 2017, no major DBMS supports SQL-92 assertions.

4.10.4 Assertions

An assertion is a named constraint that may relate to the content of individual rows of a table, to the entire contents of a table, or to a state required to exist among a number of tables.

An assertion is described by an assertion descriptor. In addition to the components of every constraint descriptor an assertion descriptor includes:

- the <search condition>.

An assertion is satisfied if and only if the specified <search condition> is not false.
DEFINING FDS IN IBM DB2

IBM DB2 supports FDs but they are limited to single attributes.

```
CREATE TABLE students (  
sid INT PRIMARY KEY,  
name VARCHAR(32),  
  CONSTRAINT student_name CHECK (name)  
  DETERMINED BY (sid)  
);
```
WHY SHOULD I CARE?

FDs seem important, but what can we actually do with them?

They allow us to decide whether a database design is correct.
→ Note that this different then the question of whether it’s a good idea for performance...
### IMPLIED DEPENDENCIES

**student(sid,cid,room,grade,name,address)**

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>room</th>
<th>grade</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-445</td>
<td>GHC 6115</td>
<td>A</td>
<td>Andy</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>456</td>
<td>15-721</td>
<td>GHC 8102</td>
<td>B</td>
<td>Tupac</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>789</td>
<td>15-445</td>
<td>GHC 6115</td>
<td>A</td>
<td>Obama</td>
<td>Chicago</td>
</tr>
<tr>
<td>012</td>
<td>15-445</td>
<td>GHC 6115</td>
<td>A</td>
<td>Waka Flocka</td>
<td>Atlanta</td>
</tr>
</tbody>
</table>

**Provided FDs**
- sid → name, address
- sid, cid → grade

**Implied FDs**
- sid, cid → grade
- sid, cid → sid
- sid, cid → cid
IMPLIED DEPENDENCIES

Given a set of FDs $\{f_1, \ldots, f_n\}$, how do we decide whether FD $g$ holds?

Compute the closure using Armstrong’s Axioms (chapter 8.4):
→ This is the set of all implied FDs.
ARMSTRONG’S AXIOMS

Reflexivity:
→ X ⊇ Y ⇒ X→Y

Augmentation:
→ X→Y ⇒ XZ→YZ

Transitivity:
→ (X→Y) ∧ (Y→Z) ⇒ X→Z

Union:
→ (X→Y) ∧ (X→Z) ⇒ X→YZ

Decomposition:
→ X→YZ ⇒ (X→Y) ∧ (X→Z)

Pseudo-transitivity:
→ (X→Y) ∧ (YW→Z) ⇒ XW→Z
Given a set $F$ of FDs $\{f_1, \ldots, f_n\}$, we define the closure $F^+$ is the set of all implied FDs.

$$\text{student}(\text{sid}, \text{cid}, \text{room}, \text{grade}, \text{name}, \text{address})$$

$$F = \begin{cases} \text{sid, cid} \rightarrow \text{grade} \\ \text{sid} \rightarrow \text{name, address} \end{cases}$$
CLOSURES

Given a set $F$ of FDs $\{f_1, \ldots, f_n\}$, we define
the closure $F^+$ is the set of all implied FDs.
WHY DO WE NEED THE CLOSURE?

With the closure we can find all FD's easily and then compute the attribute closure:

→ For a given attribute \( X \), the closure \( X^+ \) is the set of all attributes such that \( X \rightarrow A \) can be inferred using the Armstrong's Axioms.

To check if \( X \rightarrow A \):
→ Compute \( X^+ \)
→ Check if \( A \in X^+ \)
BUT AGAIN, WHY SHOULD I CARE?

Maintaining the closure at runtime is expensive:
→ The DBMS has to check all the constraints for every INSERT, UPDATE, and DELETE operation.

We want a minimal set of FDs that was enough to ensure correctness.
**CANONICAL COVER**

Given a set $F$ of FDs $\{f_1, \ldots, f_n\}$, we define the **canonical cover** $F_c$ as the minimal set of all FDs.

- $\text{sid, cid} \rightarrow \text{grade}$
- $\text{sid} \rightarrow \text{name, address}$
- $\text{sid, name} \rightarrow \text{name, address}$
- $\text{sid, cid} \rightarrow \text{grade, name}$
A canonical cover $F_c$ must have the following properties:

1. The RHS of every FD is a single attribute.
2. The closure of $F_c$ is identical to the closure of $F$ (i.e., $F_c = F$ are equivalent).
3. The $F_c$ is minimal (i.e., if we eliminate any attribute from the LHS or RHS of a FD, property #2 is violated.)
COMPUTING THE CANONICAL COVER

Given a set $F$ of FDs, examine each FD:

→ Drop extraneous LHS or RHS attributes; or redundant FDs.
→ Make sure that FDs have a single attribute in their RHS.

Repeat until no change.
COMPUTING THE CANONICAL COVER (1)

\( F: \)

- \( AB \rightarrow C \) (1)
- \( A \rightarrow BC \) (2)
- \( B \rightarrow C \) (3)
- \( A \rightarrow B \) (4)

\( F_1: \)

- \( AB \rightarrow C \) (1)
- \( A \rightarrow B \) (2')
- \( A \rightarrow C \) (2'')
- \( B \rightarrow C \) (3)
- \( A \rightarrow B \) (4)

Split (2)
COMPUTING THE CANONICAL COVER (2)

F₁:
AB → C  (1)
A → B   (2')
A → C   (2'')
B → C   (3)
A → B   (4)

Eliminate (2')

F₂:
AB → C  (1)
A → C   (2''')
B → C   (3)
A → B   (4)
COMPUTING THE CANONICAL COVER (3)

\[ F_2 : \]
\[
\begin{align*}
AB & \rightarrow C \quad (1) \\
A & \rightarrow C \quad (2'') \\
B & \rightarrow C \quad (3) \\
A & \rightarrow B \quad (4)
\end{align*}
\]

Eliminate (2'')

\[ F_3 : \]
\[
\begin{align*}
AB & \rightarrow C \quad (1) \\
B & \rightarrow C \quad (3) \\
A & \rightarrow B \quad (4)
\end{align*}
\]
COMPUTING THE CANONICAL COVER (4)

\[ F_3: \]
- \( A \rightarrow B \) (4)
- \( B \rightarrow C \) (3)
- \( A \rightarrow B \) (4)

Eliminate \( A \) from (1)

\[ F_4: \]
- \( B \rightarrow C \) (3)
- \( A \rightarrow B \) (4)
- \( B \rightarrow C \) (3)
COMPUTING THE CANONICAL COVER (5)

$F_4$:
- $B \rightarrow C$ (1')
- $B \rightarrow C$ (3)
- $A \rightarrow B$ (4)

Eliminate (1')

$F_5$:
- $B \rightarrow C$ (3)
- $A \rightarrow B$ (4)
COMPUTING THE CANONICAL COVER (5)

$F_4:$
- $B \rightarrow C$ (1')
- $B \rightarrow C$ (3)
- $A \rightarrow B$ (4)

$F_5:$
- $B \rightarrow C$ (3)
- $A \rightarrow B$ (4)

- ✓ Nothing is extraneous
- ✓ All RHS are single attributes
- ✓ Final & original set of FDs are equivalent (same closure)
NO REALLY, WHY SHOULD I CARE?

The canonical cover is the minimum number of assertions that we need to implement to make sure that our database integrity is correct.

It allows us to find the super key for a relation.
RELATIONAL MODEL: KEYS (1)

Super Key:
→ Any set of attributes in a relation that functionally determines all attributes in the relation.

Candidate Key:
→ Any super key such that the removal of any attribute leaves a set that does not functionally determine all attributes.
RELATIONAL MODEL: KEYS (2)

Super Key:
→ Set of attributes for which there are no two distinct tuples with the same values for the attributes in this set.

Candidate Key:
→ Set of attributes that uniquely identifies a tuple according to a key constraint.
RELATIONAL MODEL: KEYS (3)

Super Key:
→ A set of attributes that uniquely identifies a tuple.

Candidate Key:
→ A minimal set of attributes that uniquely identifies a tuple.

Primary Key:
→ Usually just the candidate key.
BUT WHY CARE ABOUT SUPER KEYS?

They help us determine whether it is okay to decompose a table into multiple sub-tables.

Super keys ensure that we are able to recreate the original relation through joins.
SCHEMA DECOMPOSITIONS

Split a single relation $R$ into a set of relations $\{R_1, \ldots, R_n\}$.

Not all decompositions make the database schema better:

→ Update Anomalies
→ Insert Anomalies
→ Delete Anomalies
→ Wasted Space
DECOMPOSITION GOALS

Loseless Joins
→ Want to be able to reconstruct original relation by joining smaller ones using a natural join.

Dependency Preservation
→ Want to minimize the cost of global integrity constraints based on FD’s.

Redundancy Avoidance
→ Avoid unnecessary data duplication.
**Decomposition Goals**

- **Loseless Joins**
  → Want to be able to reconstruct original relation by joining smaller ones using a natural join.

- **Dependency Preservation**
  → Want to minimize the cost of global integrity constraints based on FD’s.

- **Redundancy Avoidance**
  → Avoid unnecessary data duplication.

← **Mandatory!**

← **Nice to have, but not required**
## LOSSLESS DECOMPOSITION (1)

The provided functional dependencies (FDs) are:

- **bname** → **bcity, assets**
- **loanId** → **amt, bname**

### loans(bname, bcity, assets, cname, loanId, amt)

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Andy</td>
<td>L-17</td>
<td>$1000</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
<td>$2000</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Andy</td>
<td>L-93</td>
<td>$500</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>DJ Snake</td>
<td>L-17</td>
<td>$1000</td>
</tr>
</tbody>
</table>
**LOSSLESS DECOMPOSITION (1)**

\[ \text{loans(bname, bcity, assets, cname, loanId, amt)} \]

**Provided FDs**
- \( \text{bname} \rightarrow \text{bcity}, \text{assets} \)
- \( \text{loanId} \rightarrow \text{amt}, \text{bname} \)

**R1(bname, bcity, assets, cname)**

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>DJ Snake</td>
</tr>
</tbody>
</table>

**R2(cname, loanId, amt)**

<table>
<thead>
<tr>
<th>cname</th>
<th>loanId</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>L-17</td>
<td>$1000</td>
</tr>
<tr>
<td>Obama</td>
<td>L-23</td>
<td>$2000</td>
</tr>
<tr>
<td>Andy</td>
<td>L-93</td>
<td>$500</td>
</tr>
<tr>
<td>DJ Snake</td>
<td>L-17</td>
<td>$1000</td>
</tr>
</tbody>
</table>
**LOSSLESS DECOMPOSITION (1)**

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>DJ Snake</td>
</tr>
</tbody>
</table>

**Provided FDs**

- `bname -> bcity, assets`
- `loanId -> amt, bname`

**R1(bname, bcity, assets, cname)**

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>DJ Snake</td>
</tr>
</tbody>
</table>

**R2(cname, loanId, amt)**

<table>
<thead>
<tr>
<th>cname</th>
<th>loanId</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>L-17</td>
<td>$1000</td>
</tr>
<tr>
<td>Obama</td>
<td>L-23</td>
<td>$2000</td>
</tr>
<tr>
<td>Andy</td>
<td>L-93</td>
<td>$500</td>
</tr>
<tr>
<td>DJ Snake</td>
<td>L-17</td>
<td>$1000</td>
</tr>
</tbody>
</table>
# Lossless Decomposition (1)

**R1**(bname, bcity, assets, cname)

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>DJ Snake</td>
</tr>
</tbody>
</table>

**R2**(cname, loanId, amt)

<table>
<thead>
<tr>
<th>cname</th>
<th>loanId</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>L-17</td>
<td>$1000</td>
</tr>
<tr>
<td>Obama</td>
<td>L-23</td>
<td>$2000</td>
</tr>
<tr>
<td>Andy</td>
<td>L-93</td>
<td>$500</td>
</tr>
<tr>
<td>DJ Snake</td>
<td>L-17</td>
<td>$1000</td>
</tr>
</tbody>
</table>

Provided FDs:
- bname → bcity, assets
- loanId → amt, bname
# LOSSLESS DECOMPOSITION (2)

**R1(bname, bcity, assets, cname)**

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>DJ Snake</td>
</tr>
</tbody>
</table>

**R2(bname, loanId, amt)**

<table>
<thead>
<tr>
<th>bname</th>
<th>loanId</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-17</td>
<td>$1000</td>
</tr>
<tr>
<td>Downtown</td>
<td>L-23</td>
<td>$2000</td>
</tr>
<tr>
<td>Compton</td>
<td>L-93</td>
<td>$500</td>
</tr>
</tbody>
</table>

Provided FDs:
- `bname → bcity, assets`
- `loanId → amt, bname`
### LOSSLESS DECOMPOSITION (2)

#### R1(bname, bcity, assets, cname)

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Andy</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>DJ Snake</td>
</tr>
</tbody>
</table>

#### R2(bname, loanId, amt)

<table>
<thead>
<tr>
<th>bname</th>
<th>loanId</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-17</td>
<td>$1000</td>
</tr>
<tr>
<td>Downtown</td>
<td>L-23</td>
<td>$2000</td>
</tr>
<tr>
<td>Compton</td>
<td>L-93</td>
<td>$500</td>
</tr>
</tbody>
</table>

**Provided FDs**

- \( \text{bname} \rightarrow \text{bcity}, \text{assets} \)
- \( \text{loanId} \rightarrow \text{amt}, \text{bname} \)
LOSSLESS DECOMPOSITION (3)

R1(bname, assets, cname, loanId)

<table>
<thead>
<tr>
<th>bname</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Andy</td>
<td>L-17</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
</tr>
<tr>
<td>Compton</td>
<td>$2M</td>
<td>Andy</td>
<td>L-93</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>DJ Snake</td>
<td>L-17</td>
</tr>
</tbody>
</table>

R2(loanId, bcity, amt)

<table>
<thead>
<tr>
<th>loanId</th>
<th>bcity</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-17</td>
<td>Pittsburgh</td>
<td>$1000</td>
</tr>
<tr>
<td>L-23</td>
<td>Pittsburgh</td>
<td>$2000</td>
</tr>
<tr>
<td>L-93</td>
<td>Los Angeles</td>
<td>$500</td>
</tr>
</tbody>
</table>

Provided FDs

bname → bcity, assets
loanId → amt, bname
LOSSLESS DECOMPOSITION (3)

R1(bname, assets, cname, loanId)

<table>
<thead>
<tr>
<th>bname</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Andy</td>
<td>L-17</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
</tr>
<tr>
<td>Compton</td>
<td>$2M</td>
<td>Andy</td>
<td>L-93</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>DJ Snake</td>
<td>L-17</td>
</tr>
</tbody>
</table>

R2(loanId, bcity, amt)

<table>
<thead>
<tr>
<th>loanId</th>
<th>bcity</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-17</td>
<td>Pittsburgh</td>
<td>$1000</td>
</tr>
<tr>
<td>L-23</td>
<td>Pittsburgh</td>
<td>$2000</td>
</tr>
<tr>
<td>L-93</td>
<td>Los Angeles</td>
<td>$500</td>
</tr>
</tbody>
</table>

Provided FDs:
- bname → bcity, assets
- loanId → amt, bname

R1 → R2 and R2 → R1 are lossless.
A schema preserves dependencies if its original FDs do not span multiple tables.

Why does this matter?
→ It would be expensive to check (assuming that our DBMS supports ASSERTIONS).
**DEPENDENCY PRESERVATION (1)**

**R1(bname, assets, cname, loanId)**

<table>
<thead>
<tr>
<th>bname</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Andy</td>
<td>L-17</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
</tr>
<tr>
<td>Compton</td>
<td>$2M</td>
<td>Andy</td>
<td>L-93</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>DJ Snake</td>
<td>L-17</td>
</tr>
</tbody>
</table>

**R2(loanId, bcity, amt)**

<table>
<thead>
<tr>
<th>loanId</th>
<th>bcity</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-17</td>
<td>Pittsburgh</td>
<td>$1000</td>
</tr>
<tr>
<td>L-23</td>
<td>Pittsburgh</td>
<td>$2000</td>
</tr>
<tr>
<td>L-93</td>
<td>Los Angeles</td>
<td>$500</td>
</tr>
</tbody>
</table>

**Provided FDs**

- bname → bcity, assets
- loanId → amt, bname
### DEPENDENCY PRESERVATION (1)

**R1** \((\text{bname}, \text{assets}, \text{cname}, \text{loanId})\)

<table>
<thead>
<tr>
<th>bname</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Andy</td>
<td>L-17</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
</tr>
<tr>
<td>Compton</td>
<td>$2M</td>
<td>Andy</td>
<td>L-93</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>DJ Snake</td>
<td>L-17</td>
</tr>
</tbody>
</table>

**R2** \((\text{loanId}, \text{bcity}, \text{amt})\)

<table>
<thead>
<tr>
<th>loanId</th>
<th>bcity</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-17</td>
<td>Pittsburgh</td>
<td>$1000</td>
</tr>
<tr>
<td>L-23</td>
<td>Pittsburgh</td>
<td>$2000</td>
</tr>
<tr>
<td>L-93</td>
<td>Los Angeles</td>
<td>$500</td>
</tr>
</tbody>
</table>

**Provided FDs**

- \(\text{bname} \rightarrow \text{bcity, assets}\)
- \(\text{loanId} \rightarrow \text{amt, bname}\)
### Provided FDs

- $\text{bname} \rightarrow \text{bcity, assets}$
- $\text{loanId} \rightarrow \text{amt, bname}$

### Table R1

<table>
<thead>
<tr>
<th>bname</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Andy</td>
<td>L-17</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
</tr>
<tr>
<td>Compton</td>
<td>$2M</td>
<td>Andy</td>
<td>L-93</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>DJ Snake</td>
<td>L-17</td>
</tr>
</tbody>
</table>

### Table R2

<table>
<thead>
<tr>
<th>loanId</th>
<th>bcity</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-1</td>
<td>Pittsburgh</td>
<td>$1000</td>
</tr>
<tr>
<td>L-23</td>
<td>Pittsburgh</td>
<td>$2000</td>
</tr>
<tr>
<td>L-93</td>
<td>Los Angeles</td>
<td>$500</td>
</tr>
</tbody>
</table>
To test whether the decomposition \( R = \{R_1, \ldots, R_n\} \) preserves the FD set \( F \):

→ Compute \( F^+ \)
→ Compute \( G \) as the union of the set of FDs in \( F^+ \) that are covered by \( \{R_1, \ldots, R_n\} \)
→ Compute \( G^+ \)
→ If \( F^+ = G^+ \), then \( \{R_1, \ldots, R_n\} \) is Dependency Preserving
DEPENDENCY PRESERVATION (2)

Is $R = \{R_1, R_2\}$ dependency preserving?

$F^+ = \{A \rightarrow B, AB \rightarrow D, A \rightarrow D, C \rightarrow D\}$

$R_1(A,B,C) \quad R_2(C,D)$

$F = \{A \rightarrow B, \ AB \rightarrow D, \ C \rightarrow D\}$
DEPENDENCY PRESERVATION (2)

Is \( R = \{R_1, R_2\} \) dependency preserving?

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

\[
G = \{A \rightarrow B\} \cup \{C \rightarrow D\}
\]

FDs covered by \( R_1 \)  
FDs covered by \( R_2 \)

\[
R_1(A, B, C) \quad R_2(C, D)
\]

\[
F = \{A \rightarrow B, AB \rightarrow D, C \rightarrow D\}
\]
DEPENDENCY PRESERVATION (2)

Is $R = \{R_1, R_2\}$ dependency preserving?

$F^+ = \{A \rightarrow B, AB \rightarrow D, A \rightarrow D, C \rightarrow D\}$

$G = \{A \rightarrow B\} \cup \{C \rightarrow D\}$

$G^+ = \{A \rightarrow B, C \rightarrow D\}$

$R_1(A, B, C)$ $R_2(C, D)$

$F = \{A \rightarrow B, AB \rightarrow D, C \rightarrow D\}$
**DEPENDENCY PRESERVATION (2)**

Is $R = \{R_1, R_2\}$ dependency preserving?  

$F^+ = \{A \rightarrow B, AB \rightarrow D, A \rightarrow D, C \rightarrow D\}$

$G = \{A \rightarrow B\} \cup \{C \rightarrow D\}$

$G^+ = \{A \rightarrow B, C \rightarrow D\}$

$F^+ \neq G^+$ because $(A \rightarrow D) \in (F^+ - G^+)$

$(A \rightarrow D) \notin G^+$

---

$R_1(A, B, C)$  
$R_2(C, D)$

$F = \{A \rightarrow B, AB \rightarrow D, C \rightarrow D\}$

*Decomposition is not DP*
Is $R = \{R_1, R_2\}$ dependency preserving?

$R_1(A,B,D)$  $R_2(C,D)$

$F = \{A \rightarrow B, \ AB \rightarrow D, \ C \rightarrow D\}$
DEPENDENCY PRESERVATION (3)

Is $R = \{R_1, R_2\}$ dependency preserving?

$F^+ = \{A \rightarrow B, AB \rightarrow D, A \rightarrow D, C \rightarrow D\}$

$G = \{A \rightarrow B, A \rightarrow D, AB \rightarrow D\} \cup \{C \rightarrow D\}$

$G^+ = \{A \rightarrow B, AB \rightarrow D, A \rightarrow D, C \rightarrow D\}$

$F^+ = G^+$

$F = \{A \rightarrow B, AB \rightarrow D, C \rightarrow D\}$

$R_1(A, B, D) \ R_2(C, D)$

Decomposition is DP
REDUNDANCY AVOIDANCE

We want to avoid duplicate entries in a relation for a FD.

When there exists some FD $X \rightarrow Y$ covered by relation and $X$ is not a super key.
DECOMPOSITION SUMMARY

Lossless Joins

→ Motivation: Avoid information loss.
→ Goal: No noise introduced when reconstituting universal relation via joins.
→ Test: At each decomposition
  \[ R = (R_1 \cup R_2), \text{ check whether } (R_1 \cap R_2) \Rightarrow R_1 \text{ or } (R_1 \cap R_2) \Rightarrow R_2. \]
Dependency Preservation

→ Motivation: Efficient FD assertions.
→ Goal: No global integrity constraints that require joins of more than one table with itself.
→ Test: \( R = (R_1 \cup \ldots \cup R_n) \) is dependency preserving if closure of FD’s covered by each \( R_1 = \text{closure of FD’s covered by } R = F. \)
CONCLUSION

Functional dependencies are simple to understand.

They will allow us to reason about schema decompositions.
PROJECT #1

You will build the first component of your storage manager.
→ Extendible Hash Table
→ LRU Replacement Policy
→ Buffer Pool Manager

All of the projects are based on SQLite, but you will not be able to use your storage manager just yet after this first project.

Due Date:
Monday Oct 2\textsuperscript{nd} @ 11:59pm
TASK #1 – EXTENDIBLE HASH TABLE

Build a thread-safe extendible hash table.
→ Use unordered buckets to store key/value pairs.
→ You must support growing table size.
→ You do not need to support shrinking.

General Hints:
→ You can use `std::hash` and `std::mutex`. 
**TASK #2 – LRU REPLACEMENT POLICY**

Build a data structure that tracks the usage of Page objects in the buffer pool using the least-recently used policy.

**General Hints:**
→ Your LRUReplacer does not need to worry about the "pinned" status of a Page.
TASK #3 – BUFFER POOL MANAGER

Combine your hash table and LRU replacer together to manage the allocation of pages.
→ Need to maintain an internal data structures of allocated + free pages.
→ We will provide you components to read/write data from disk.

General Hints:
→ Make sure you get the order of operations correct when pinning.
GETTING STARTED

Download the source code from the project webpage.

Make sure you can build it on your machine.
→ We've test it on Andrew machines, OSX, and Linux.
→ It should compile on Windows 10 w/ Ubuntu, but we haven't tried it.
THINGS TO NOTE

Do not change any file other than the six that you have to hand it.

The projects are cumulative.

We will not be providing solutions.

Post your questions on Canvas or come to TA office hours.
    → We will not help you debug.
Your project implementation must be your own work.
→ You may **not** copy source code from other groups or the web.
→ Do **not** publish your implementation on Github.

Plagiarism will **not** be tolerated. See [CMU's Policy on Academic Integrity](#) for additional information.
NEXT CLASS

Normal Forms