Carnegie Mellon University

Functional Dependencies

Lecture #04

Database Systems
15-445/15-645
Fall 2017

Andy Pavlo
Computer Science Dept.
Carnegie Mellon Univ.

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

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

sid	cid	room	grade	name	address
123	15-445	GHC 6115	A	Andy	Pittsburgh
456	15-721	GHC 8102	B	Tupac	Los Angeles
789	15-445	GHC 6115	A	Obama	Chicago
012	15-445	GHC 6115	C	Waka Flocka	Atlanta
789	15-721	GHC 8102	A	Obama	Chicago

EXAMPLE DATABASE

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

sid	cid	room	grade	name	address
123	15-445	GHC 6115	A	Andy	Pittsburgh
456	15-721	GHC 8102	B	Tupac	Los Angeles
789	15-445	GHC 6115	A	Obama	Chicago
012	15-445	GHC 6115	C	Waka Flocka	Atlanta
789	15-721	GHC 8102	A	Obama	Chicago

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)

sid	name	address
123	Andy	Pittsburgh
456	Tupac	Los Angeles
789	Obama	Chicago
012	Waka Flocka	Atlanta

courses(sid,cid,grade)

sid	cid	grade
123	15-415	A
456	15-721	B
789	15-415	A
012	15-415	C
789	15-721	A

rooms(cid,room)

cid	room
15-415	GHC 6115
15-721	GHC 8102

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→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.

R1(sid,name,address)

sid	name	address
123	Andy	Pittsburgh
456	Tupac	Los Angeles
789	Obama	Chicago
012	Waka Flocka	Atlanta

FUNCTIONAL DEPENDENCIES

Formal Definition:

→ X→Y ⇒ $(t_1[x]=t_2[x] \Rightarrow t_1[y]=t_2[y])$

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

R1(sid,name,address)

sid	name	address
123	Andy	Pittsburgh
456	Tupac	Los Angeles
789	Obama	Chicago
012	Waka Flocka	Atlanta

X Y

✓ sid→name

CARNEGIE MELLON DATABASE GROUP

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.

R1(sid,name,address)

sid	name	address
123	Andy	Pittsburgh
456	Tupac	Los Angeles
789	Obama	Chicago
012	Waka Flocka	Atlanta
555	Andy	Providence

??? name→address

FUNCTIONAL DEPENDENCIES

Two FDs **X→Y** and **X→Z** can be written in shorthand as **X→YZ**.

But **XY→Z** is not the same as the two FDs **X→Z** and **Y→Z**.

DEFINING FDS IN SQL (EX.1)

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

FD_1: sid → name

SQL-92

```
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))
```


CARNEGIE MELLON DATABASE GROUP

DEFINING FDS IN SQL (EX.2)

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

FD_1: sid → name
FD_2: sid → address

SQL-92

```
CREATE ASSERTION student-name-addr
 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)))
```

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.

CARNEGIE MELLON DATABASE GROUP

DEFINING FDS IN IBM DB2

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

FD_1: sid → name

IBM DB2

```
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)

sid	cid	room	grade	name	address
123	15-445	GHC 6115	A	Andy	Pittsburgh
456	15-721	GHC 8102	B	Tupac	Los Angeles
789	15-445	GHC 6115	A	Obama	Chicago
012	15-445	GHC 6115	A	Waka Flocka	Atlanta

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:
→ $X \supseteq Y \Rightarrow X \rightarrow Y$

Augmentation:
→ $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$

Transitivity:
→ $(X \rightarrow Y) \land (Y \rightarrow Z) \Rightarrow X \rightarrow Z$

Union:
→ $(X \rightarrow Y) \land (X \rightarrow Z) \Rightarrow X \rightarrow YZ$

Decomposition:
→ $X \rightarrow YZ \Rightarrow (X \rightarrow Y) \land (X \rightarrow Z)$

Pseudo-transitivity:
→ $(X \rightarrow Y) \land (YW \rightarrow Z) \Rightarrow XW \rightarrow Z$

CLOSURES

Given a set **F** of FDs $\{f_1,\ldots,f_n\}$, we define the closure **F+** is the set of all implied FDs.

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

F:

```
sid,cid → grade
sid → name,address
```

CLOSURES

Given a set F of FDs {f_1,…,f_n}, we define the closure F+ is the set of all implied FDs.

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

F+:

F:
- `sid,cid → grade`
- `sid → name,address`

Reflexivity → `sid,name → sid`

Decomposition → `sid → name`

Decomposition → `sid → address`

Augmentation → `sid,grade → name,grade`

Transitivity → `sid,cid → grade,name`

⋮

WHY DO WE NEED THE CLOSURE?

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

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

To check if X→A:

→ Compute X+
→ Check if A∈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.

F:

F_c:

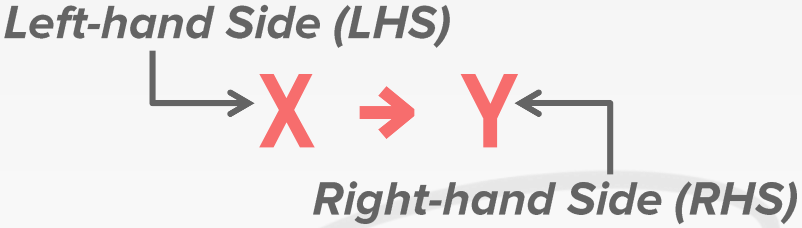
```
sid,cid → grade
sid → name,address
```

```
sid,name → name,address
sid,cid → grade,name
```

CANONICAL COVER DEFINITION

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.

Left-hand Side (LHS)

$X \rightarrow Y$

Right-hand Side (RHS)

COMPUTING THE CANONICAL COVER (1)

F:

AB → C (1)
A → BC (2)
B → C (3)
A → B (4)

Split (2)

F_1:

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

COMPUTING THE CANONICAL COVER (2)

F_1:

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

Eliminate (2')

F_2:

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

COMPUTING THE CANONICAL COVER (3)

F_2:

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

Eliminate (2'')

F_3:

AB → C (1)
B → C (3)
A → B (4)

COMPUTING THE CANONICAL COVER (4)

F_3:

AB → C	(1)	
B → C	(3)	
A → B	(4)	

Eliminate A from (1)

→

F_4:

B → C	(1')
B → C	(3)
A → B	(4)

COMPUTING THE CANONICAL COVER (5)

F_4:

B → C	(1')
B → C	(3)
A → B	(4)

Eliminate (1') →

F_5:

B → C	(3)
A → B	(4)

COMPUTING THE CANONICAL COVER (5)

F_4:

B → C (1')

B → C (3)

A → B (4)

Eliminate (1')

F_c

F_5:

B → C (3)

A → 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.

← **Mandatory!**

Dependency Preservation

→ Want to minimize the cost of global integrity constraints based on FD's.

Redundancy Avoidance

→ Avoid unnecessary data duplication.

← **Nice to have, but not required**

LOSSLESS DECOMPOSITION (1)

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

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

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Andy	L-17	$1000
Downtown	Pittsburgh	$9M	Obama	L-23	$2000
Compton	Los Angeles	$2M	Andy	L-93	$500
Downtown	Pittsburgh	$9M	DJ Snake	L-17	$1000

LOSSLESS DECOMPOSITION (1)

Provided FDs

bname → bcity,assets

loanId → amt,bname

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

R1(bname,bcity,assets,cname)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Andy
Downtown	Pittsburgh	$9M	Obama
Compton	Los Angeles	$2M	Andy
Downtown	Pittsburgh	$9M	DJ Snake

R2(cname,loanId,amt)

cname	loanId	amt
Andy	L-17	$1000
Obama	L-23	$2000
Andy	L-93	$500
DJ Snake	L-17	$1000

LOSSLESS DECOMPOSITION (1)

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

R1(bname,bcity,assets,cname)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Andy
Downtown	Pittsburgh	$9M	Obama
Compton	Los Angeles	$2M	Andy
Downtown	Pittsburgh	$9M	DJ Snake

⋈

R2(cname,loanId,amt)

cname	loanId	amt
Andy	L-17	$1000
Obama	L-23	$2000
Andy	L-93	$500
DJ Snake	L-17	$1000

LOSSLESS DECOMPOSITION (1)

Provided FDs

bname → bcity,assets

loanId → amt,bname

R1(bname,bcity,assets,cname)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Andy
Downtown	Pittsburgh	$9M	Obama
Compton	Los Angeles	$2M	Andy
Downtown	Pittsburgh	$9M	DJ Snake

$\bowtie$

R2(cname,loanId,amt)

cname	loanId	amt
Andy	L-17	$1000
Obama	L-23	$2000
Andy	L-93	$500
DJ Snake	L-17	$1000

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Andy	L-17	$1000
Downtown	Pittsburgh	$9M	Andy	L-93	$500
Downtown	Pittsburgh	$9M	Obama	L-23	$2000
Compton	Los Angeles	$2M	Andy	L-17	$1000
Compton	Los Angeles	$2M	Andy	L-93	$500
Downtown	Pittsburgh	$9M	DJ Snake	L-17	$1000

LOSSLESS DECOMPOSITION (2)

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

R1(bname,bcity,assets,cname)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Andy
Downtown	Pittsburgh	$9M	Obama
Compton	Los Angeles	$2M	Andy
Downtown	Pittsburgh	$9M	DJ Snake

⋈

R2(bname,loanId,amt)

bname	loanId	amt
Downtown	L-17	$1000
Downtown	L-23	$2000
Compton	L-93	$500

LOSSLESS DECOMPOSITION (2)

Provided FDs

bname → bcity,assets

loanId → amt,bname

R1(bname,bcity,assets,cname)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Andy
Downtown	Pittsburgh	$9M	Obama
Compton	Los Angeles	$2M	Andy
Downtown	Pittsburgh	$9M	DJ Snake

⋈

R2(bname,loanId,amt)

bname	loanId	amt
Downtown	L-17	$1000
Downtown	L-23	$2000
Compton	L-93	$500

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Andy	L-17	$1000
Downtown	Pittsburgh	$9M	Andy	L-23	$2000
Downtown	Pittsburgh	$9M	Obama	L-17	$1000
Downtown	Pittsburgh	$9M	Obama	L-23	$2000
Compton	Los Angeles	$2M	Andy	L-93	$500
Downtown	Pittsburgh	$9M	DJ Snake	L-23	$1000

LOSSLESS DECOMPOSITION (3)

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

R1(bname,assets,cname,loanId)

bname	assets	cname	loanId
Downtown	$9M	Andy	L-17
Downtown	$9M	Obama	L-23
Compton	$2M	Andy	L-93
Downtown	$9M	DJ Snake	L-17

⋈

R2(loanId,bcity,amt)

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

LOSSLESS DECOMPOSITION (3)

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

R1(bname,assets,cname,loanId)

bname	assets	cname	loanId
Downtown	$9M	Andy	L-17
Downtown	$9M	Obama	L-23
Compton	$2M	Andy	L-93
Downtown	$9M	DJ Snake	L-17

⋈

R2(loanId,bcity,amt)

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

✓

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Andy	L-17	$1000
Downtown	Pittsburgh	$9M	Obama	L-23	$2000
Compton	Los Angeles	$2M	Andy	L-93	$500
Downtown	Pittsburgh	$9M	DJ Snake	L-17	$1000

DEPENDENCY PRESERVATION

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)

bname	assets	cname	loanId
Downtown	$9M	Andy	L-17
Downtown	$9M	Obama	L-23
Compton	$2M	Andy	L-93
Downtown	$9M	DJ Snake	L-17

R2(loanId,bcity,amt)

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

Provided FDs

bname → bcity,assets

loanId → amt,bname

DEPENDENCY PRESERVATION (1)

R1(bname,assets,cname,loanId)

bname	assets	cname	loanId
Downtown	$9M	Andy	L-17
Downtown	$9M	Obama	L-23
Compton	$2M	Andy	L-93
Downtown	$9M	DJ Snake	L-17

R2(loanId,bcity,amt)

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

Provided FDs

bname → bcity,assets

loanId → amt,bname

DEPENDENCY PRESERVATION (1)

R1(bname,assets,cname,loanId)

bname	assets	cname	loanId
Downtown	$9M	Andy	L-17
Downtown	$9M	Obama	L-23
Compton	$2M	Andy	L-93
Downtown	$9M	DJ Snake	L-17

R2(loanId,bcity,amt)

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

Provided FDs

bname → bcity,assets

loanId → amt,bname

DEPENDENCY PRESERVATION

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→B,AB→D,A→D,C→D}

```
R1(A,B,C) R2(C,D)
F = {A→B, AB→D, C→D}
```


CARNEGIE MELLON
DATABASE GROUP

DEPENDENCY PRESERVATION (2)

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

F+ = {A→B, AB→D, A→D, C→D}

G = {A→B} U {C→D}

FDs covered by R_1 | FDs covered by R_2

R1(A,B,C) R2(C,D)
F = {A→B, AB→D, C→D}

DEPENDENCY PRESERVATION (2)

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

F+ = {A→B,AB→D,A→D,C→D}

G = {A→B} U {C→D}

G+ = {A→B,C→D}

R1(A,B,C) R2(C,D)

F = {A→B, AB→D, C→D}

DEPENDENCY PRESERVATION (2)

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

$(A \rightarrow D) \in F+$

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

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

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

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

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

R1(A,B,C) R2(C,D)

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

Decomposition is not DP

DEPENDENCY PRESERVATION (3)

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

```
R1(A,B,D) R2(C,D)
F = {A→B, AB→D, C→D}
```

DEPENDENCY PRESERVATION (3)

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

F+ = {A→B, AB→D, A→D, C→D}

G = {A→B, A→D, AB→D} U {C→D}

G+ = {A→B, AB→D, A→D, C→D}

F+ = G+

R1(A,B,D) R2(C,D)

F = {A→B, AB→D, 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→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 $R=(R_1 \cup R_2)$, check whether $(R_1 \cap R_2) \rightarrow R_1$ or $(R_1 \cap R_2) \rightarrow R2$.

CARNEGIE MELLON DATABASE GROUP

DECOMPOSITION SUMMARY

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 = 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 manger just yet after this first project.

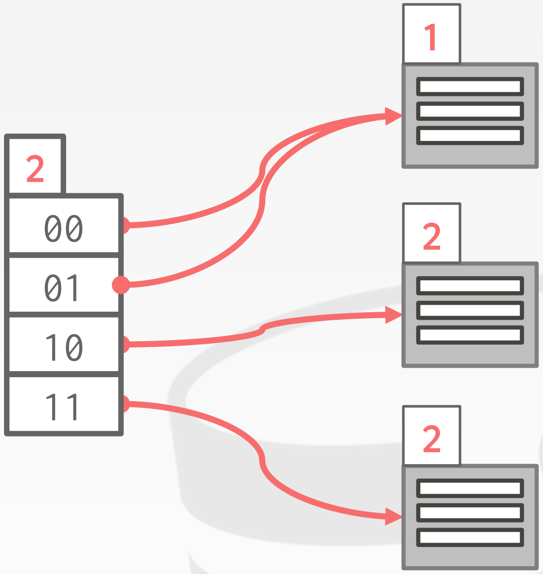
SQLite

Due Date:
Monday Oct 2nd @ 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`.

CARNEGIE MELLON DATABASE GROUP

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

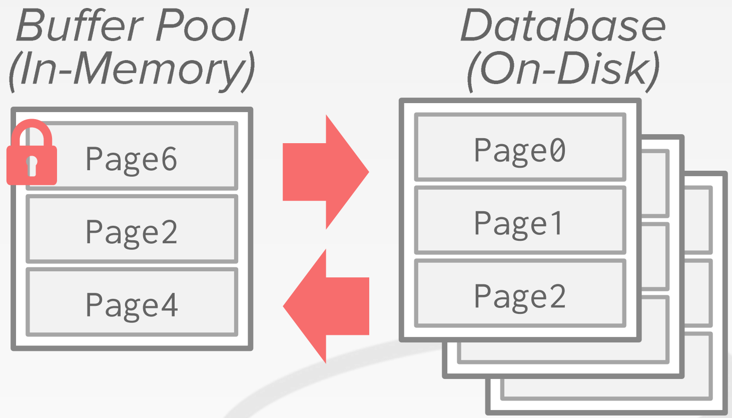
Combine your hash table and LRU replacer together to manage the allocation of pages.

→ 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.



CARNEGIE MELLON DATABASE GROUP

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.

PLAGIARISM WARNING

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