# Design Theory and Normalization

CSC343
Mark Kazakevich
(with slides from Diane Horton, Jeff Ullman)

# What's the point?

- Most likey, you have some idea of how to make a schema for a data set
- But how do you know it's a good schema?
- How do you make it better?
- Is there an 'optimal' schema for your data?

# Database ‘Design’ Theory

- Improve your (probably bad) schema systematically
- Try to get a schema that is in a **“normal form”** that guarantees **good properties**.
  - “Normal” in the sense of conforming to a standard.
- The process of converting a schema to a normal form is called ***normalization***.

# Database ‘Design’ Theory

- **General idea:**
  - Express *constraints* on the relationships between attributes
  - Use these constraints to *decompose* the relations

# What are these ‘constraints’?

- We’ve defined constraints before like keys, foreign keys, etc. based on our knowledge of the **problem domain**.
- We can define other domain-specific constraints on the data that can help us design better schemas
  - These constraints are called **functional dependencies**

- A **functional dependency** (FD) on a relation R is a statement of the form:

  If two tuples of R **agree** on all of the attributes

  $$A_1, A_2, \ldots, A_n$$

  then they must also **agree** on all of another list of attributes

  $$B_1, B_2, \ldots, B_m$$

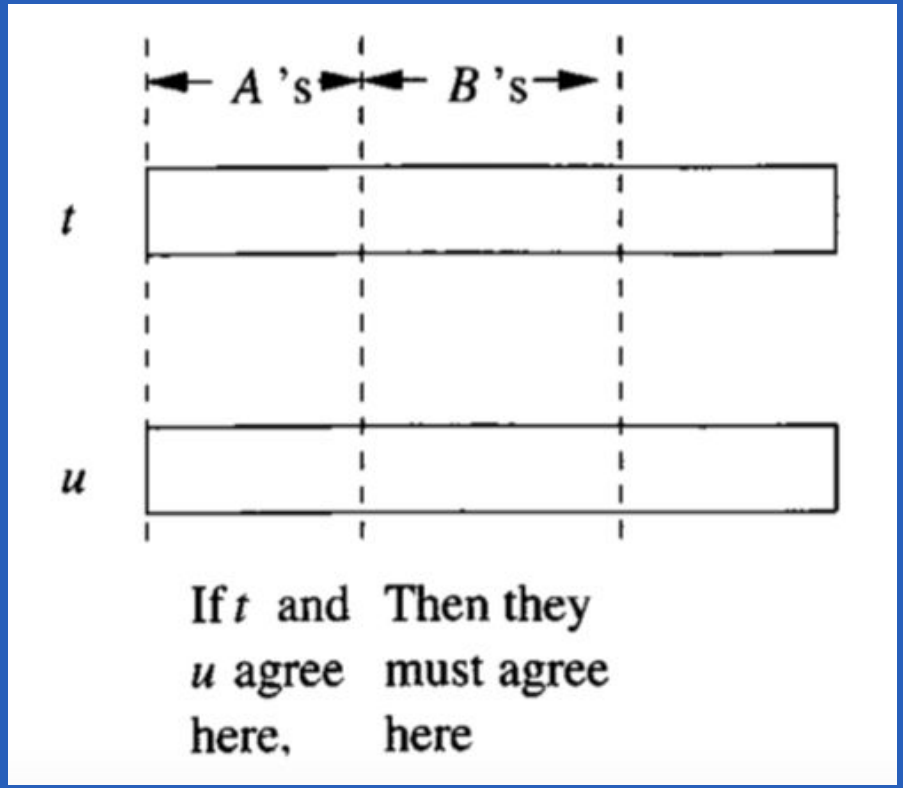
- We write this FD formally as

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

and we say:

"$A_1, A_2, \ldots, A_n$ functionally determines $B_1, B_2, \ldots, B_m$"

# Visual representation




**Note:** it is not a requirement that the attributes B have to follow A
directly - they can be anywhere

# Example

| *title* | *year* | *length* | *genre* | *studioName* | *starName* |
|---|---|---|---|---|---|
| Star Wars | 1977 | 124 | SciFi | Fox | Carrie Fisher |
| Star Wars | 1977 | 124 | SciFi | Fox | Mark Hamill |
| Star Wars | 1977 | 124 | SciFi | Fox | Harrison Ford |
| Gone With the Wind | 1939 | 231 | drama | MGM | Vivien Leigh |
| Wayne's World | 1992 | 95 | comedy | Paramount | Dana Carvey |
| Wayne's World | 1992 | 95 | comedy | Paramount | Mike Meyers |

What is a functional dependency here?
(think about the domain)

# Example

| *title* | *year* | *length* | *genre* | *studioName* | *starName* |
|---|---|---|---|---|---|
| Star Wars | 1977 | 124 | SciFi | Fox | Carrie Fisher |
| Star Wars | 1977 | 124 | SciFi | Fox | Mark Hamill |
| Star Wars | 1977 | 124 | SciFi | Fox | Harrison Ford |
| Gone With the Wind | 1939 | 231 | drama | MGM | Vivien Leigh |
| Wayne's World | 1992 | 95 | comedy | Paramount | Dana Carvey |
| Wayne's World | 1992 | 95 | comedy | Paramount | Mike Meyers |

```
title year -> length genre studioName
```

*What assumptions did we make?*

G-M, Ullman, Widom

# Example

| *title* | *year* | *length* | *genre* | *studioName* | *starName* |
|---|---|---|---|---|---|
| Star Wars | 1977 | 124 | SciFi | Fox | Carrie Fisher |
| Star Wars | 1977 | 124 | SciFi | Fox | Mark Hamill |
| Star Wars | 1977 | 124 | SciFi | Fox | Harrison Ford |
| Gone With the Wind | 1939 | 231 | drama | MGM | Vivien Leigh |
| Wayne's World | 1992 | 95 | comedy | Paramount | Dana Carvey |
| Wayne's World | 1992 | 95 | comedy | Paramount | Mike Meyers |

What is **not** a functional dependency here?

# Coincidence or FD?

- An FD is an assertion about **every** instance of the relation.
- You can't know it holds just by looking at one instance.
- You must use **knowledge of the domain** to determine whether an FD holds.

# FDs are closely related to **keys**

- Suppose K is a set of attributes for relation R.
- Our old definition of **superkey**:
  - *a set of attributes for which no two rows can have the same values.*
- Using FDs
  - K is a **superkey** for R **iff** K functionally determines **all of R**.

# FDs are a generalization of keys

- key:


- Functional dependency:


- An FD can be more subtle.

# FD Properties

# Trivial FDs

- Suppose `A1,A2...An -> B1,B2...Bn` holds
- Trivial FDs are those where the right side is a subset of the left side, that is,

$$\{B1,B2...Bn\} \subseteq \{A1,A2...An\}$$

Examples:

- `title -> title`
- `title year -> title`
- `title year -> {}`

# Splitting Rules

- An FD such as `A -> BCD` can be split up into a set of **equivalent** FDs:
  - `A -> B`
  - `A -> C`
  - `A -> D`
- We are ‘splitting’ the Right Hand Side of the FD (can also combine the three into the original)
  - Can we split the left hand side?
    - No.

# Inferring FDs

- Given a set of FDs, we can often **infer** further FDs.
- This will be handy when we apply FDs to the problem of database design.
- Big task: given a set of FDs,
  - infer every other FD that must also hold.
- Simpler task: given a set of FDs,
  - check whether a given FD must also hold.

# What we'd like to be able to find out:

- If `A -> B` and `B -> C` hold,
  must `A -> C` hold?
- If `A -> H`, `C -> F`, and `FG -> AD` hold,
  must `FA -> D` hold?
  must `CG -> FH` hold?
- If `H -> GD`, `HD -> CE`, and `BD -> A` hold,
  must `EH -> C` hold?
- Note: we are not generating new FDs,
  but **testing** a specific possible one.

# How do we figure them out?

- **Method 1** (first principles):
  - Refer to FDs you know and use definition of functional dependency
  - This can get complicated
- **Method 2**
  - The **Closure Test**
  - Systematic, and much easier!

# Closure { }+

- Suppose $\{A_1, A_2, \ldots, A_n\}$ is a set of attributes and S is a set of FDs
- The **closure** of $\{A_1, A_2, \ldots, A_n\}$ under the FD's in S is the set of attributes B such that:
  - Every relation that satisfies all the FD's in set S also satisfies $A_1A_2..A_n$ -> B.
- We denote the closure of a set of attributes $A_1A_2..A_n$ by $\{A_1, A_2, ..A_n\}$+.

# Closure Algorithm

- How do we find $\mathbf{Y^+}$, the closure of a set of attributes Y under a set of FDs S.

```
Attribute_closure(Y, S):
    Initialize Y+ to Y; split RHS's of FDs if necessary
    Repeat until no more changes occur:
        If there is an FD LHS->RHS in S
        such that LHS is in Y+:
            Add RHS to Y+
Return Y+
```

# Example

- Relation **R**(A, B, C, D, E, F)
- FDs: AB->C, BC->AD, D->E, CF->B
- What is **{A, B}+** (the closure of {A, B})

X = {A,B}; split BC->AD to BC->A, BC->D
LHS of AB->C in X, so X={A,B,C}
LHS of BC->A and BC->D in X, so X={A,B,C,D}
LHS of D->E in X, so X={A,B,C,D,E}
(can not use CF->B,since F not in X}
So we're done, {A, B}+ = {A,B,C,D,E}

# Closure Test

- S is a set of FDs; LHS -> RHS is a single FD.
  - Return true iff LHS -> RHS follows from S.

```
FD_follows(S, LHS  RHS):
  Y+ = Attribute_closure(LHS, S)
  return (RHS is in Y+)
```

# Property that follows:

- Transitive Rule
  - (`A -> B` and `B -> C`) implies `A -> C`
  - Check this using closure test
    - C would appear in $A^+$

# Projecting FDs

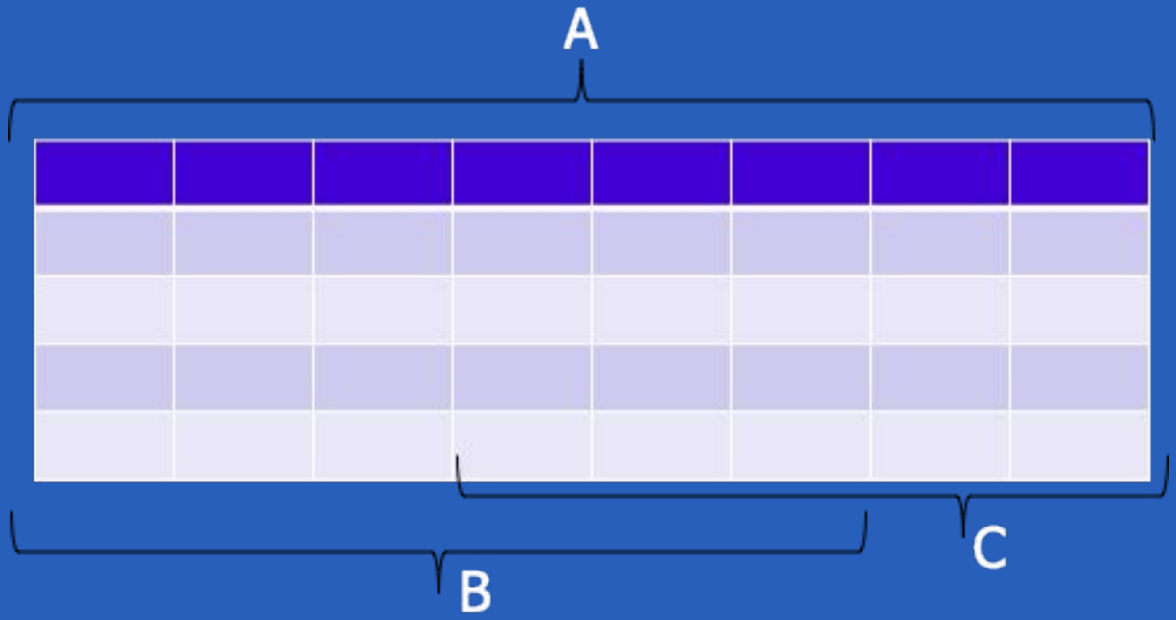
# Projecting FDs

- Later, we will learn how to ***normalize*** a schema by ***decomposing*** relations into smaller ones
  - This is the whole point of this theory.
- We will need to know what FDs hold in the new, smaller relations.
  - We must **project** our FDs onto the attributes of our new relations.

# Projecting FDs

Relation **R**(A1..An), Set of attributes: A

Sets of attributes B and C such that B ∪ C = A



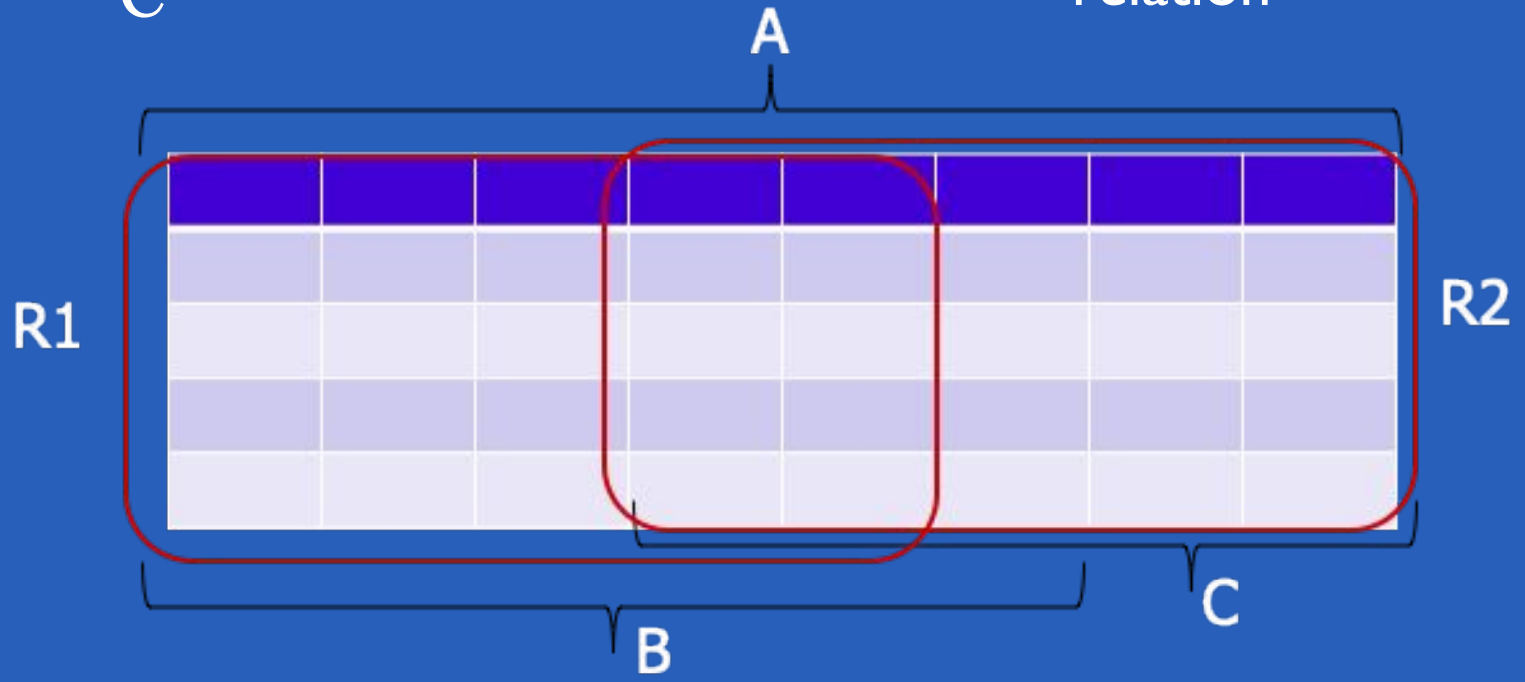
# Projecting FDs

$R1 = \pi_B(R)$

$R2 = \pi_C(R)$

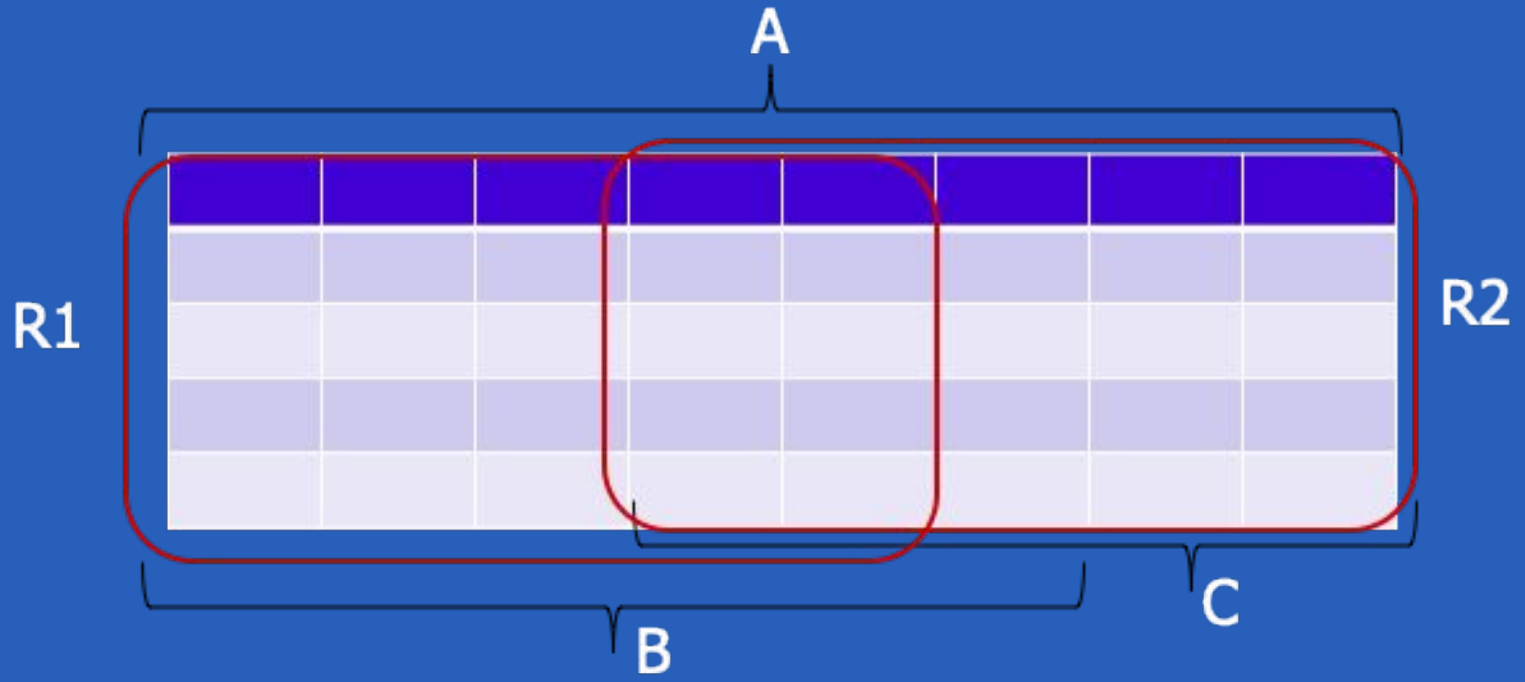
$R1 \bowtie R2 = R$

**Question:** What FDs hold in the new relation



# Projecting FDs

**Question:** If a set of FDs S hold in R,
what FDs hold in the new relations R1 and R2?



# Projection Algorithm

- **S** is a set of FDs; **L** is a set of attributes.
- Return the projection of **S** onto **L**:
  all FDs that follow from **S** and involve only attributes from **L**.

```
Project(S, L):
    Initialize T to {}.
    For each subset X of L:
        Compute X+    # Close X and see what we get.
    For every attribute A in X+:
        If A is in L: # X -> A is only relevant if A is in
                      # L (we know X is).
            add X  A to T.
    Return T.
```

# Some algorithm speed-ups

- No need to add `X->A` if A is in X itself.
  It's a trivial FD.
- These subsets of X won't yield anything, so no need to compute their closures:
  - the empty set
  - the set of all attributes in X

Neither are big savings, but …

# A big speed-up

- If we find $X^+$ = all attributes, we can ignore any superset of X.
- It can only give use "weaker" FDs (with more on the LHS).

This is a big time saver!

# Minimal basis

- We saw earlier that we can very often rewrite sets of FDs in equivalent ways.
  - Example: `S1 = {A -> BC}` is equivalent to `S2 = {A->B, A->C}`.
- Given a set of FDs S, we may want to find a **minimal basis:** A set of FDs that is equivalent, but has
  - no redundant FDs, and
  - no FDs with unnecessary attributes on the LHS.
  - All RHS will be single attributes

# Minimal Basis

- S is a set of FDs. Return a **minimal basis** for S.

```
Minimal_basis(S):
    1. Split the RHS of each FD
    2. For each FD X->Y where |X| ≥ 2:
          If you can remove an attribute from X
          (use closures!) and get an FD that
          follows from S:
              Do so!  (It's a stronger FD.)
    3. For each FD f :
          If S – {f} implies f (use closures!) :
              Remove f from S.
```

# Some comments on Minimal Basis

- Often there are multiple possible results.
  - Depends on the order in which you consider the possible simplifications.
- After you identify a redundant FD,
  you must not use it when computing subsequent closures.

# and less intuitive..

When you are computing closures to decide whether the LHS of an FD

X -> Y

can be simplified, continue to use that FD.