UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

CS411 - SQL: Triggers, Views, and Indexes

illinois.edu

Announcements

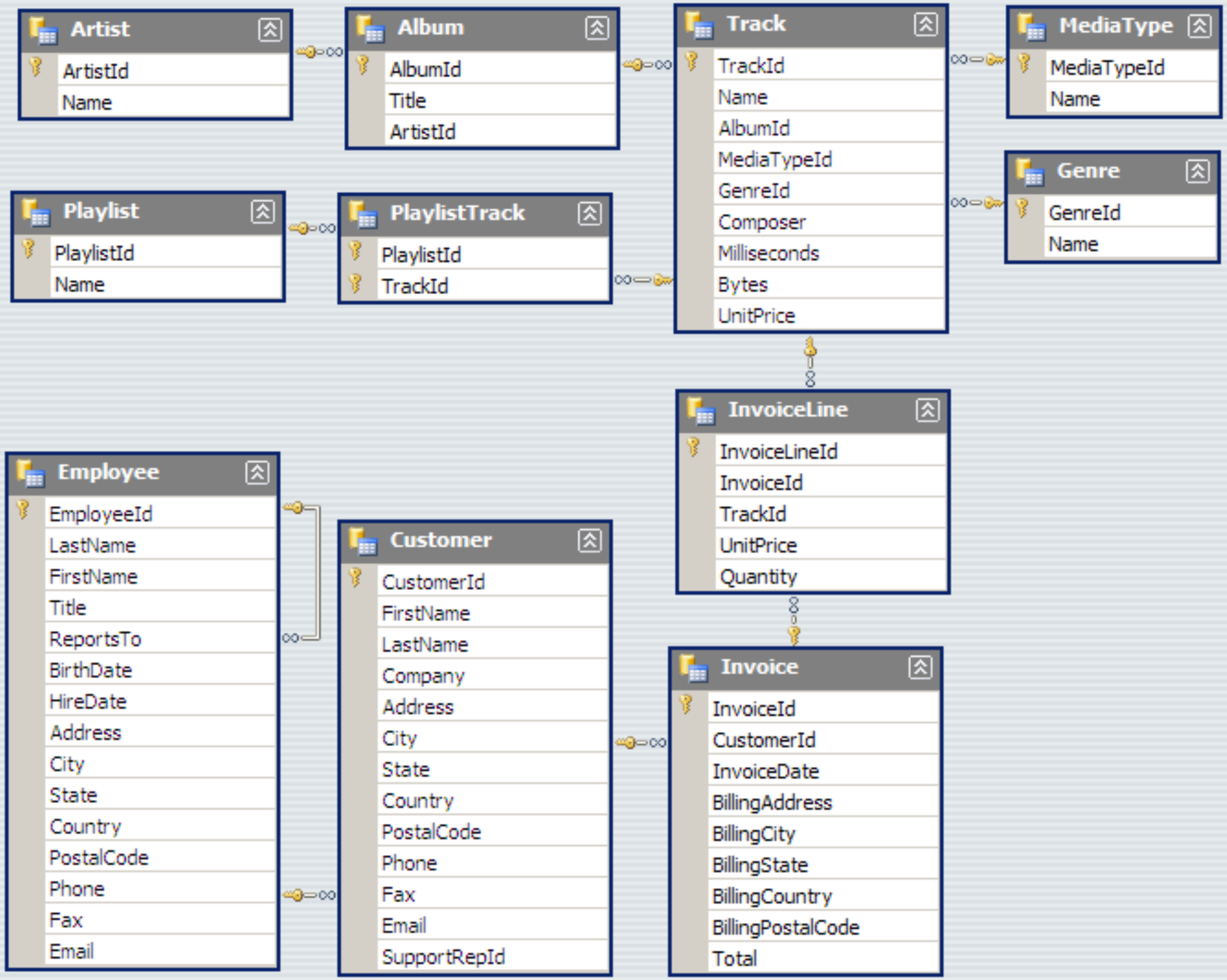
- Project Track 1
  - Stage 2 due Today
  - Stage 3 due Monday
- HW2 due Tomorrow night
  - NO HAND WRITTEN ASSIGNMENTS!
- MP1 resubmission will be available
  - if you do it, no extra credit points!

Announcements

- Midterm 1 March 1st
- Ungraded SQL assignment available
  - prepare for exam
  - prepare for MP2
  - help for project

Review

- What parts of ACID do ***transactions*** help guarantee?
- What parts of ACID do ***constraints*** guarantee?
- What were the three types of read phenomena we learned about?
- What types of constraints did we learn?


Artist
ArtistId
Name
Album
AlbumId
Title
ArtistId
Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice
MediaType
MediaTypeId
Name
Genre
GenreId
Name
Playlist
PlaylistId
Name
PlaylistTrack
PlaylistId
TrackId
InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity
Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email
Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId
Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

Chinook

- http://chinookdatabase.codeplex.com
- Open source sample database
  - provide scripts to import data into MySQL, sqlite, and many more
  - Based on an digital media store
  - Data is simulated or based on iTunes data

Setup Demo

Harsh Realities

- SQL implementations differ wildly
  - MySQL
    - doesn’t bother with CHECK constraints
    - doesn’t implement INTERSECT
  - sqlite
    - doesn’t implement OUTER JOIN
    - can’t add constraints to an existing table

Harsh Realities

- SQL syntax is different, too
  - in sqlite this is allowed:

    ```
    SELECT *
    FROM R, S, T
    ON (R.a = S.b AND S.b = T.c);
    ```
  - MySQL has AUTO_INCREMENT
    sqlite has AUTOINCREMENT

Harsh Realities

- That's just MySQL and sqlite
  - SQLServer, Oracle, PostgreSQL, ...
- Moral of the story:
  - Just pick one and stick with it
  - You'll have to translate what you learn to your DBMS of choice

Constraints

- Add attribute constraint to Employee
  - force email address to conform to a pattern
- ALTER TABLE doesn’t work for CHECK
  - We need to create a new table
  - Copy in the values
  - Drop old table
  - Rename new table

Constraints

- I'll do all this with a transaction
- Here's what we're going to change:

```
Email NVARCHAR(60)
=>
Email NVARCHAR(60)
    CHECK (email LIKE "_%@_%.com")
```

Constraints Demo

Triggers

- Constraints are
  - inefficient to implement
  - not very flexible
  - not always implemented
- Triggers enable a dynamic response to an event

Triggers

1. Awakened by an *event*
   – Insertions, deletions, updates
   – Transaction COMMITs
2. Test a *condition*
3. Perform an *action*
   – Any SQL statement is allowed

Sometimes called "ECA rules"

Trigger Syntax

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
    {BEFORE|AFTER} {INSERT|DELETE|UPDATE}
        ON <table_name>
    [REFERENCING
        [NEW AS <new_row_name>] [OLD AS <old_row_name>]]
    [FOR EACH ROW [WHEN (<trigger_condition>)]]
    <trigger_body>
```

Trigger Syntax

- COMPLICATED!
- If you want to write your own, read reference material first
- Let's look at examples
  - See why/how they're used
  - Learn a bit of the syntax

Example

```
CREATE TRIGGER NoFallenJedi
AFTER UPDATE OF side ON Jedi
REFERENCING
   OLD ROW AS oldR,
   NEW ROW AS newR
FOR EACH ROW
WHEN oldR.side=“Light” AND newR.side=“Dark”
   UPDATE Jedi
   SET side=oldR.side
   WHERE name=newR.name
```

illinois.edu

Example

```
CREATE TRIGGER NoFallenJedi
AFTER UPDATE OF side ON Jedi
REFERENCING
   OLD ROW AS oldR,
   NEW ROW AS newR
FOR EACH ROW
WHEN oldR.side=“Light” AND newR.side=“Dark”
   UPDATE Jedi
   SET side=oldR.side
   WHERE name=newR.name
```

illinois.edu

Example

```
CREATE TRIGGER ForceBalance
AFTER UPDATE OF side ON Jedi
REFERENCING
    OLD TABLE AS oldT,
    NEW TABLE AS newT
FOR EACH STATEMENT
WHEN (SELECT COUNT(*) FROM Jedi WHERE side='Light')=(SELECT COUNT(*) FROM JEDI WHERE side='Dark')
BEGIN
    DELETE FROM Jedi
    WHERE (name) IN (SELECT name FROM newT);
    INSERT INTO Jedi (SELECT * FROM OldT);
END;
```

Example

```
CREATE TRIGGER DefaultGreenSaber
BEFORE INSERT ON Jedi
REFERENCING
   NEW ROW AS newR,
   NEW TABLE AS newT
FOR EACH ROW
WHEN newR.saberCol IS NULL
   UPDATE newT SET saberCol=“Green”
```

Trigger Demo

Views

- Subqueries can be used a lot
- ***Views*** are a way to query them like tables
- Two types
  - Virtual
    - do not exist physically
    - queried again each time
  - Materialized
    - physically stored on disk

View

- Syntax

```
CREATE VIEW Villians AS
   SELECT * FROM JEDI
   WHERE side=“Dark”;
```

Updateable Views

- Some views can be updated
  - Require certain circumstances:
    1. No subqueries in WHERE clause
    2. Only one relation in FROM clause
    3. Any attributes outside of the view must allow NULL
- If we want to update other views, we should use an INSTEAD OF trigger

View Demo

illinois.edu

Materialized Views

- If we use a view a lot, we can physically realize it on the disk
- Pros:
  - Faster to execute queries
- Cons:
  - More disk usage
  - Updates to underlying tables slower

Materialized Views

- Syntax

```
CREATE MATERIALIZED VIEW
Villians AS
   SELECT * FROM JEDI
   WHERE side=“Dark”;
```

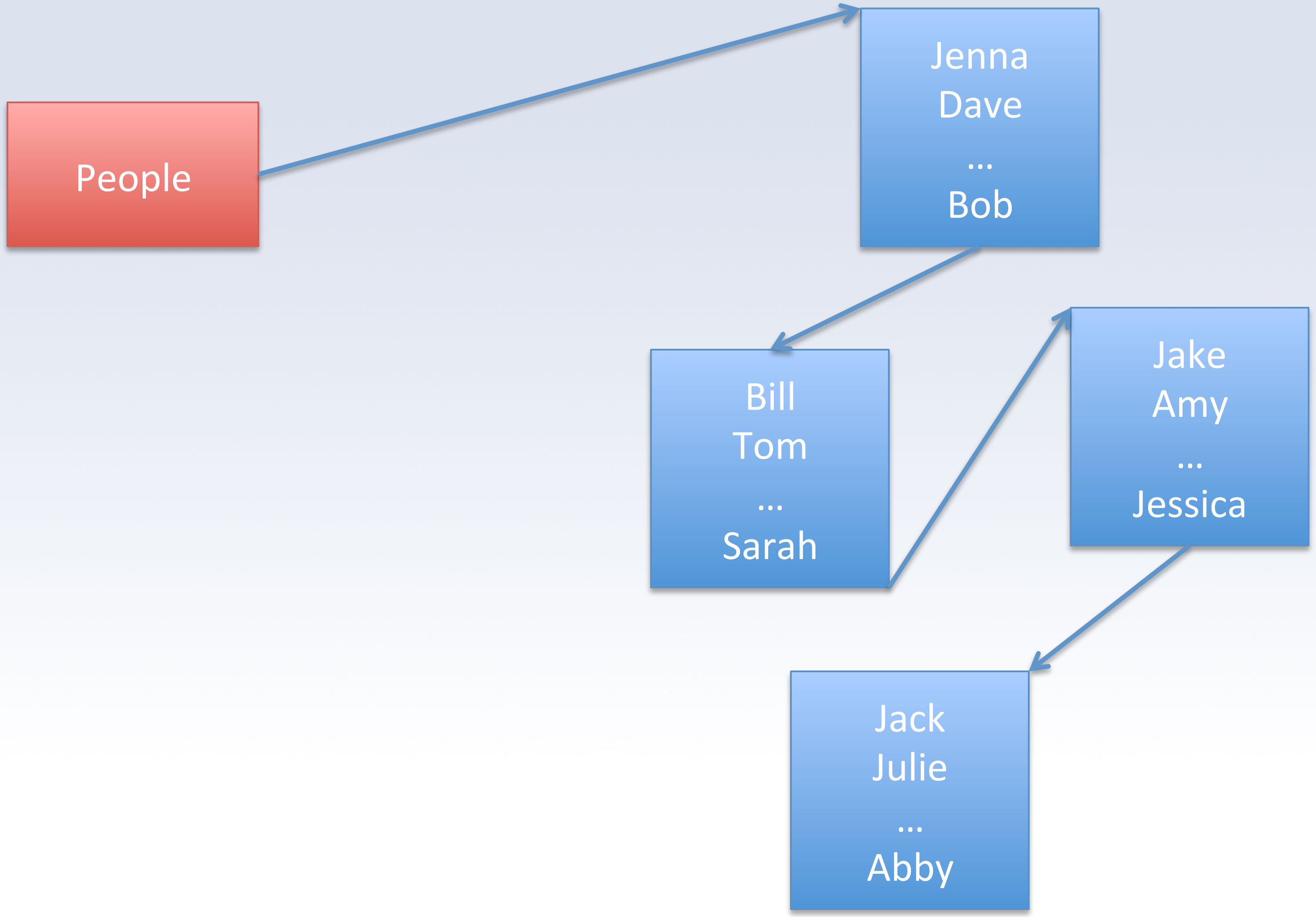
Materialized Views

- Used in enterprise scale DBMS implementations like Oracle and SQL Server
- Not supported in MySQL or sqlite

Indexes

- In reality, our tables are scattered across the disk
- Searching for specific tuples is inefficient

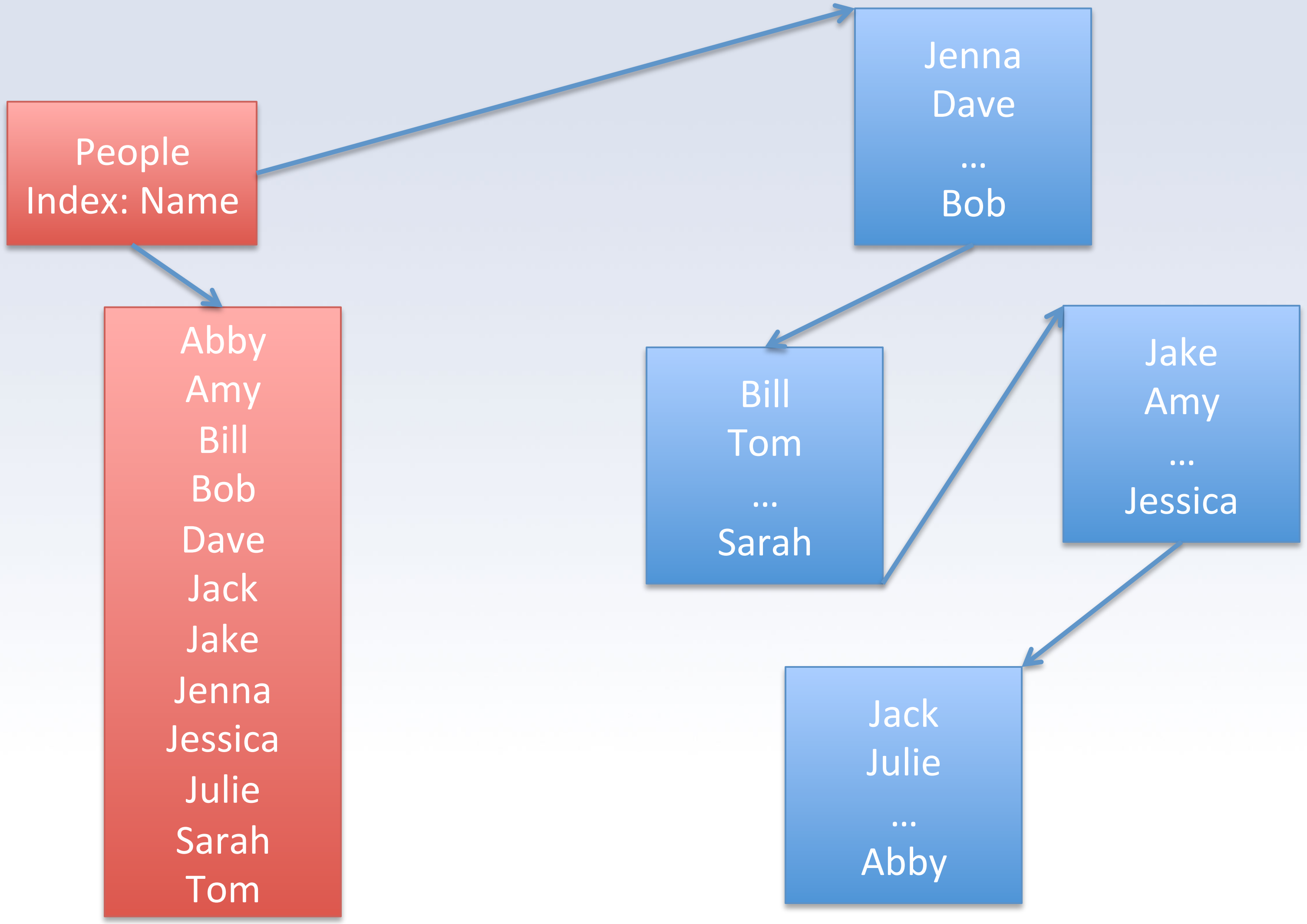
Indexes


People
Jenna
Dave
…
Bob
Bill
Tom
…
Sarah
Jake
Amy
…
Jessica
Jack
Julie
…
Abby

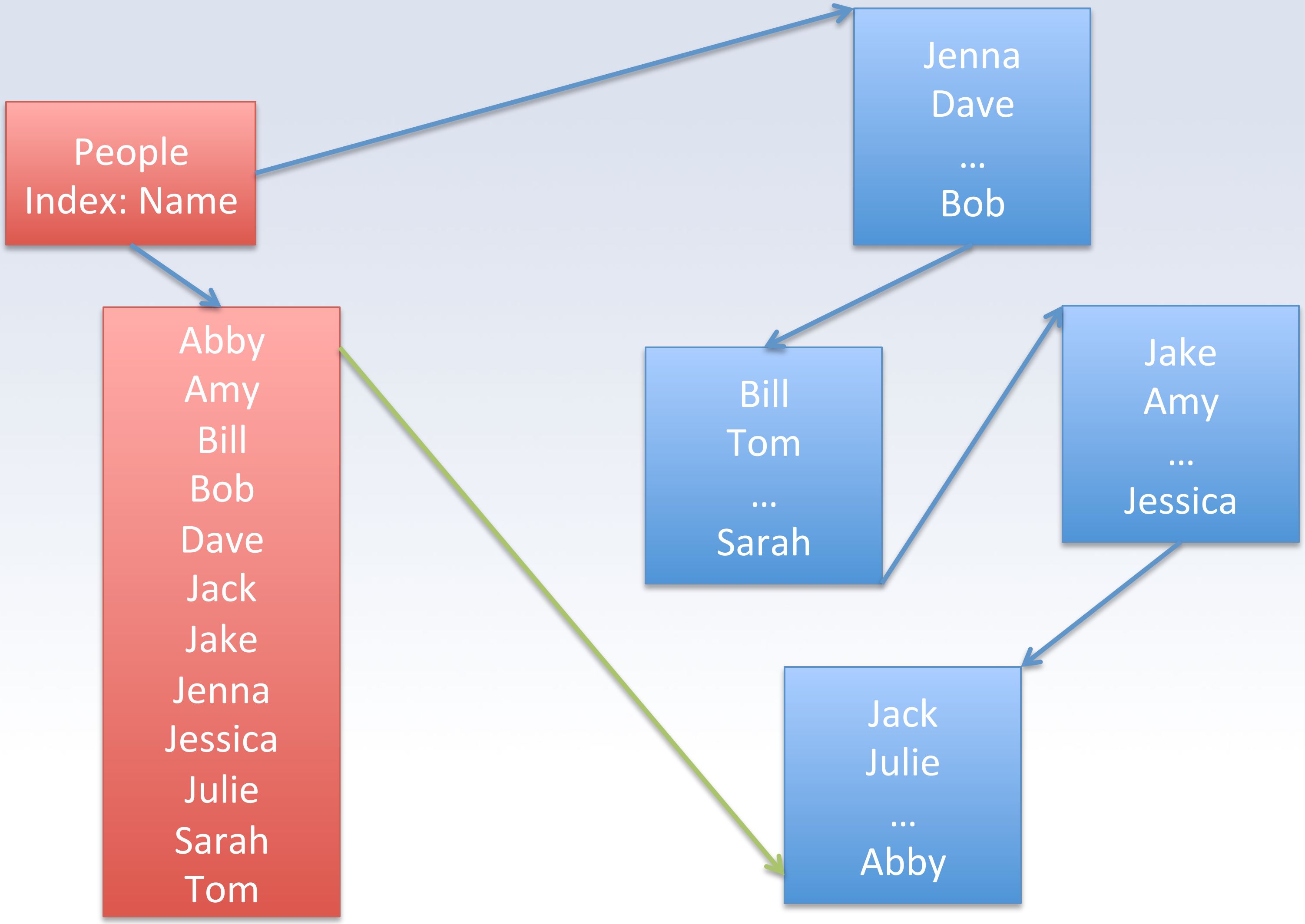
Indexes

- We can speed this process up using by ***indexing*** our table
- An ***index*** is a searchable structure that maps a series of attributes to actual tuples
- Queries using these attributes will execute faster

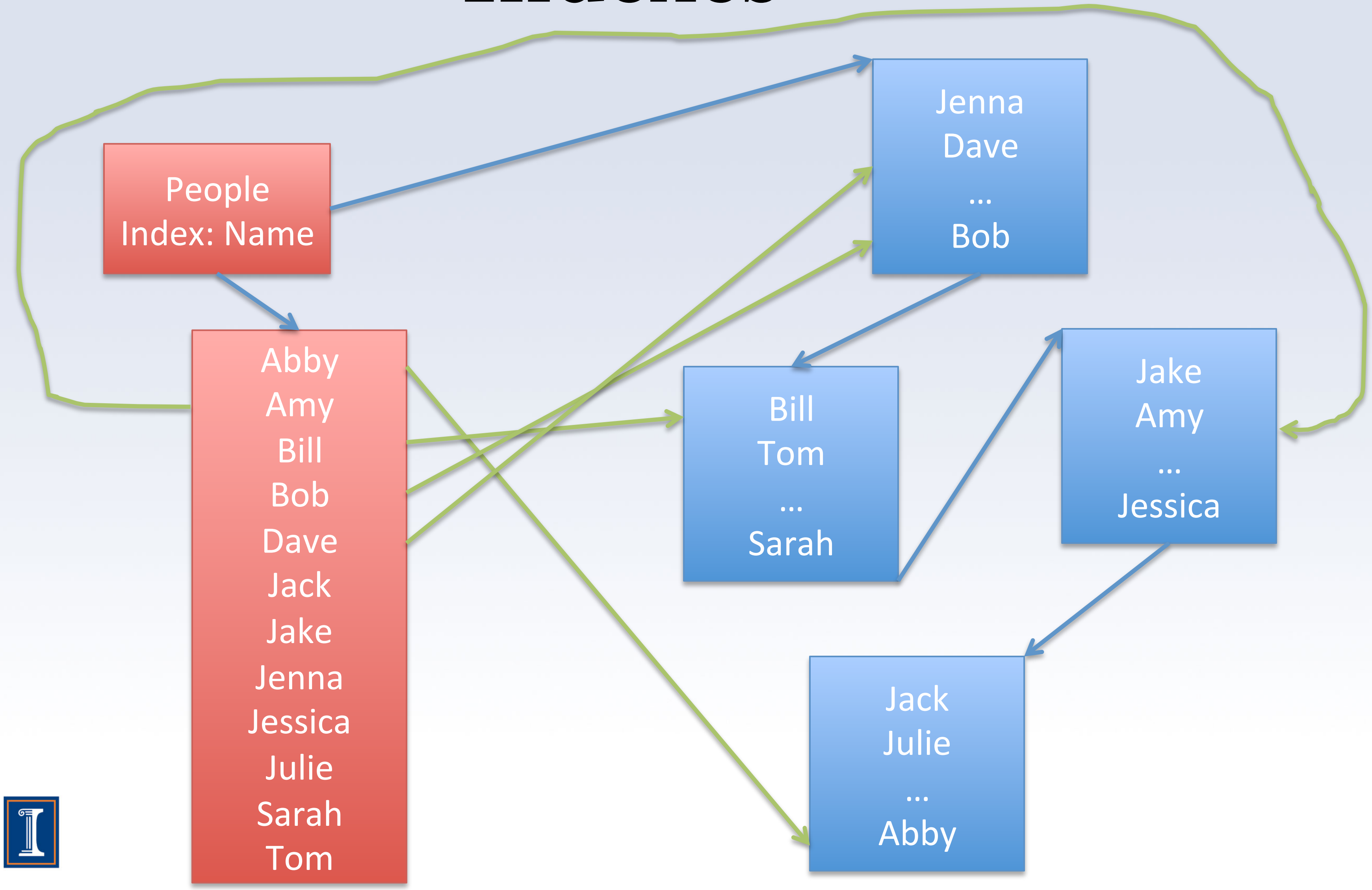
Indexes


People
Index: Name
Abby
Amy
Bill
Bob
Dave
Jack
Jake
Jenna
Jessica
Julie
Sarah
Tom
Jenna
Dave
…
Bob
Bill
Tom
…
Sarah
Jake
Amy
…
Jessica
Jack
Julie
…
Abby

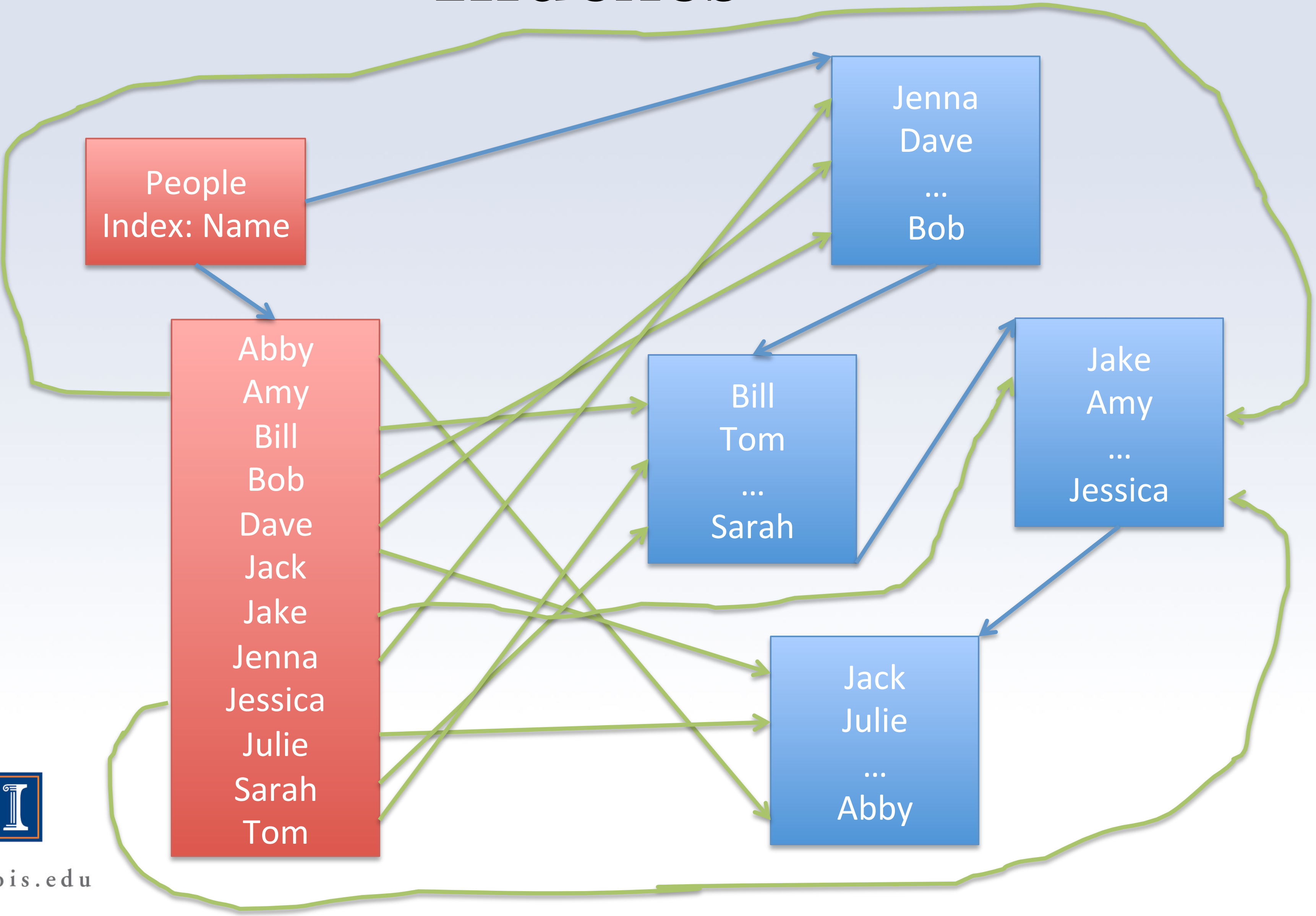
Indexes


People
Index: Name
Abby
Amy
Bill
Bob
Dave
Jack
Jake
Jenna
Jessica
Julie
Sarah
Tom
Jenna
Dave
…
Bob
Bill
Tom
…
Sarah
Jake
Amy
…
Jessica
Jack
Julie
…
Abby

Indexes


People
Index: Name
Abby
Amy
Bill
Bob
Dave
Jack
Jake
Jenna
Jessica
Julie
Sarah
Tom
Jenna
Dave
…
Bob
Bill
Tom
…
Sarah
Jake
Amy
…
Jessica
Jack
Julie
…
Abby

Indexes


People
Index: Name
Abby
Amy
Bill
Bob
Dave
Jack
Jake
Jenna
Jessica
Julie
Sarah
Tom
Jenna
Dave
…
Bob
Bill
Tom
…
Sarah
Jake
Amy
…
Jessica
Jack
Julie
…
Abby

Syntax

- We can create indexes in SQL as follows:

```
CREATE INDEX <IndexName>
ON <Table(attributes)>
```

Indexes

- Pros:
  - Speed up queries
- Cons:
  - Require disk space
  - Slow modification
- We'll investigate this in depth later

Indexes

- Multiple attributes can form an index
- Values for index don’t have to be unique
  - Attributes indexed don’t need to form a key

Indexes

- When designing indexes, we should:
  - Index on keys or “nearly” keys
  - Index on attributes we query a lot
  - Put more commonly queried attributes first
- These are just heuristics.
- A deeper understanding will come when we study implementation.

Index Demo