CMSC424: Database Design
Introduction
Relational Model

Instructor: Amol Deshpande
amol@cs.umd.edu

Today

- Wrap-up Introduction
- Current Industry Outlook
- Computing Environment
- Relational Model
- No laptop use allowed in the class !!

Some To-Dos

- Sign up for Piazza !
- Set up the computing environment (project0), and make sure you can run Vagrant+VirtualBox, PostgreSQL, IPython, etc.
- Upcoming: Reading Homework 1, Project 1: SQL

DBMSs to the Rescue

- Massively successful for *highly structured data*
  - Why ? Structure in the data (if any) can be exploited for ease of use and efficiency
  - How ?
  - Two Key Concepts:
    - Data Modeling: Allows reasoning about the data at a high level
      - e.g. “emails” have “sender”, “receiver”, “…”
      - Once we can describe the data, we can start “querying” it
    - Data Abstraction/Independence:
      - Layer the system so that the users/applications are insulated from the low-level details

DBMSs to the Rescue: Data Modeling

- Data modeling
  - **Data model**: A collection of concepts that describes how data is represented and accessed
  - **Schema:** A description of a specific collection of data, using a given data model

  - Some examples of data models that we will see
    - Relational, Entity-relationship model, XML. JSON…
    - Object-oriented, object-relational, semantic data model, RDF…

  - Why so many models ?
    - Tension between descriptive power and ease of use/efficiency
    - More powerful models → more data can be represented
    - More powerful models → harder to use, to query, and less efficient

DBMSs to the Rescue: Data Abstraction

- Probably *the* most important purpose of a DBMS
- Goal: Hiding *low-level details* from the users of the system
  - Alternatively: the principle that
    - *applications and users should be insulated from how data is structured and stored*
  - Also called *data independence*

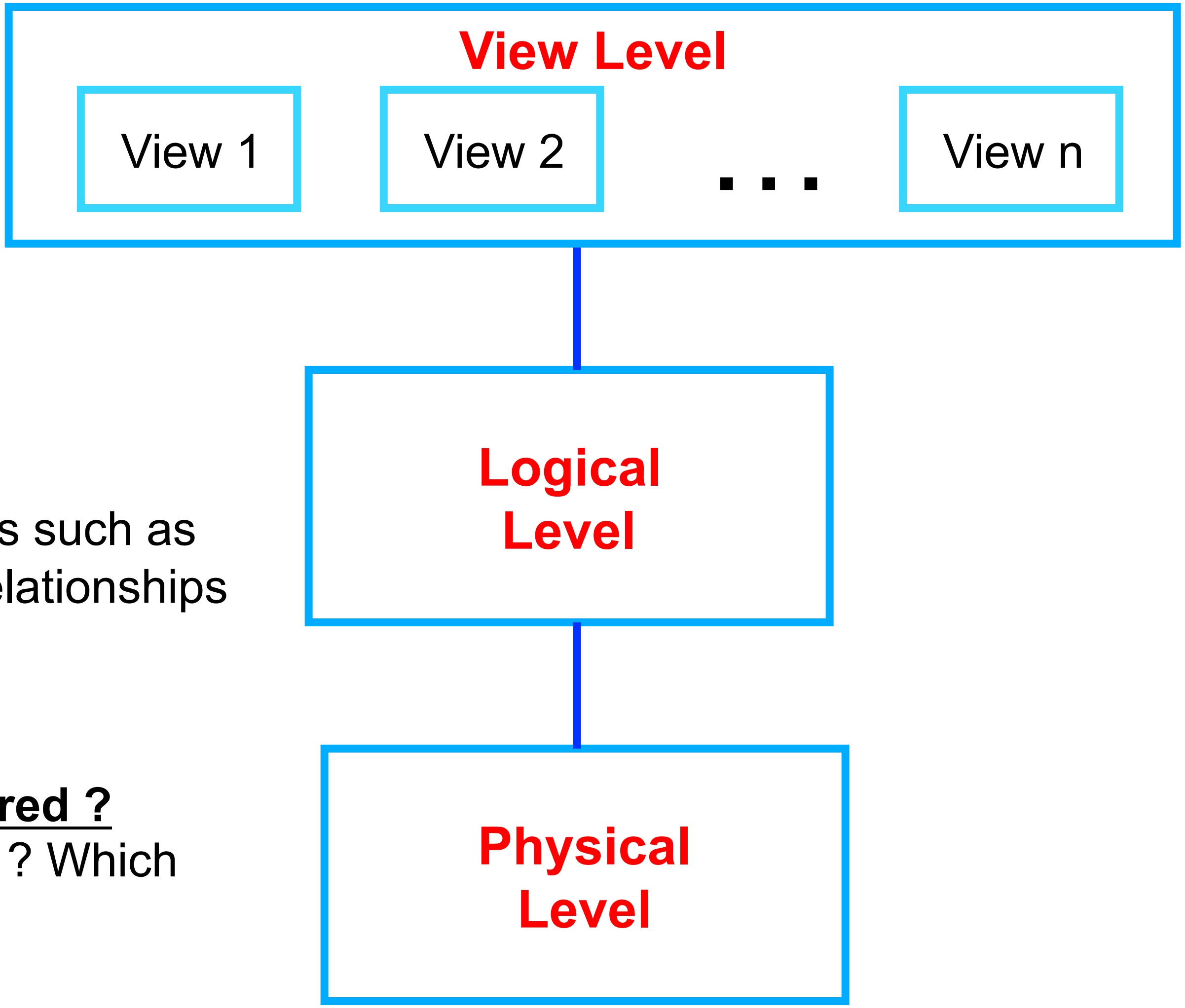
- Through use of *logical abstractions*

Data Abstraction

What data users and application programs see ?

What data is stored ?
describe data properties such as data semantics, data relationships

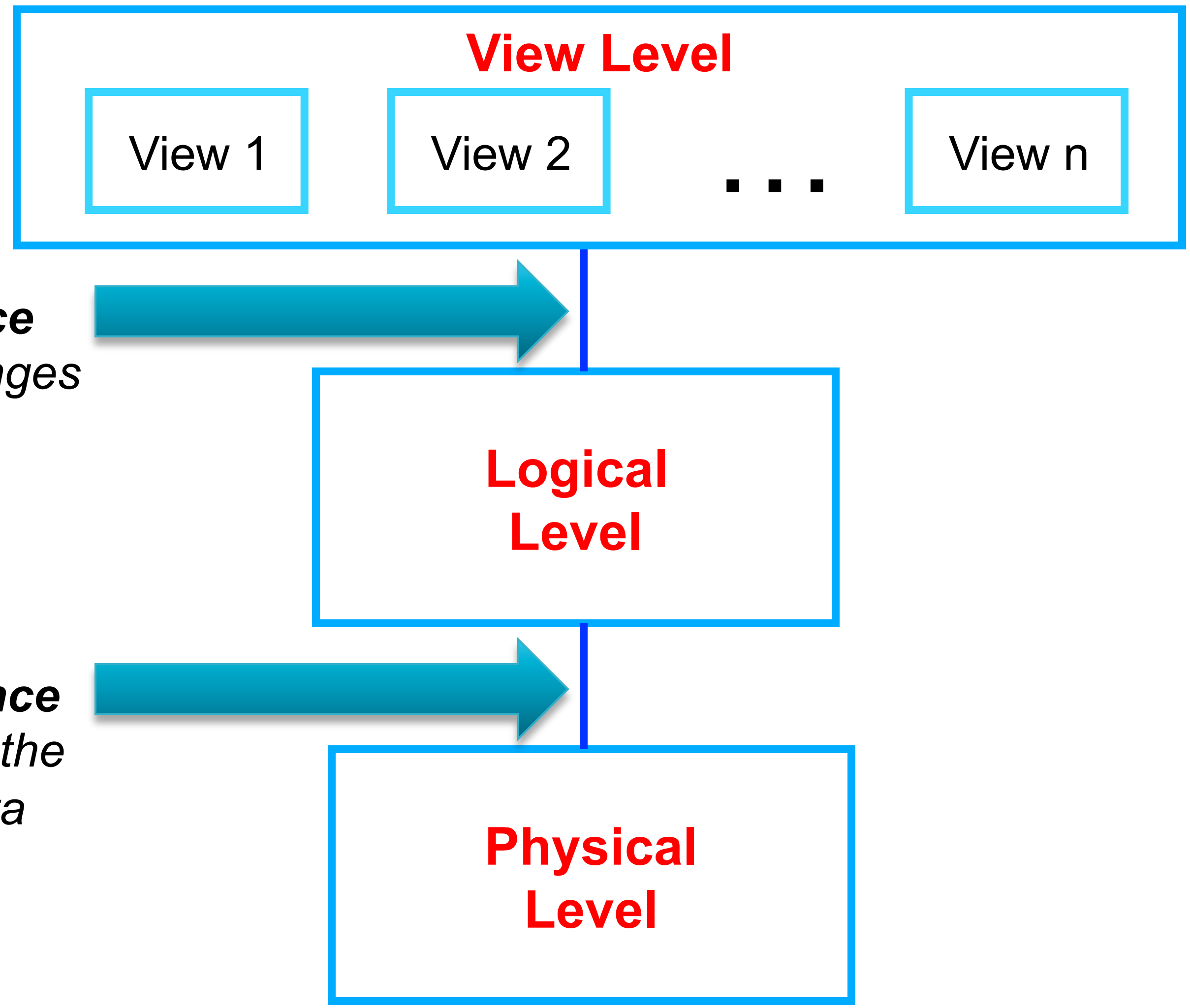
How data is actually stored ?
e.g. are we using disks ? Which file system ?



Data Abstraction

Logical Data Independence
Protection from logical changes to the schema

Physical Data Independence
Protection from changes to the physical structure of the data

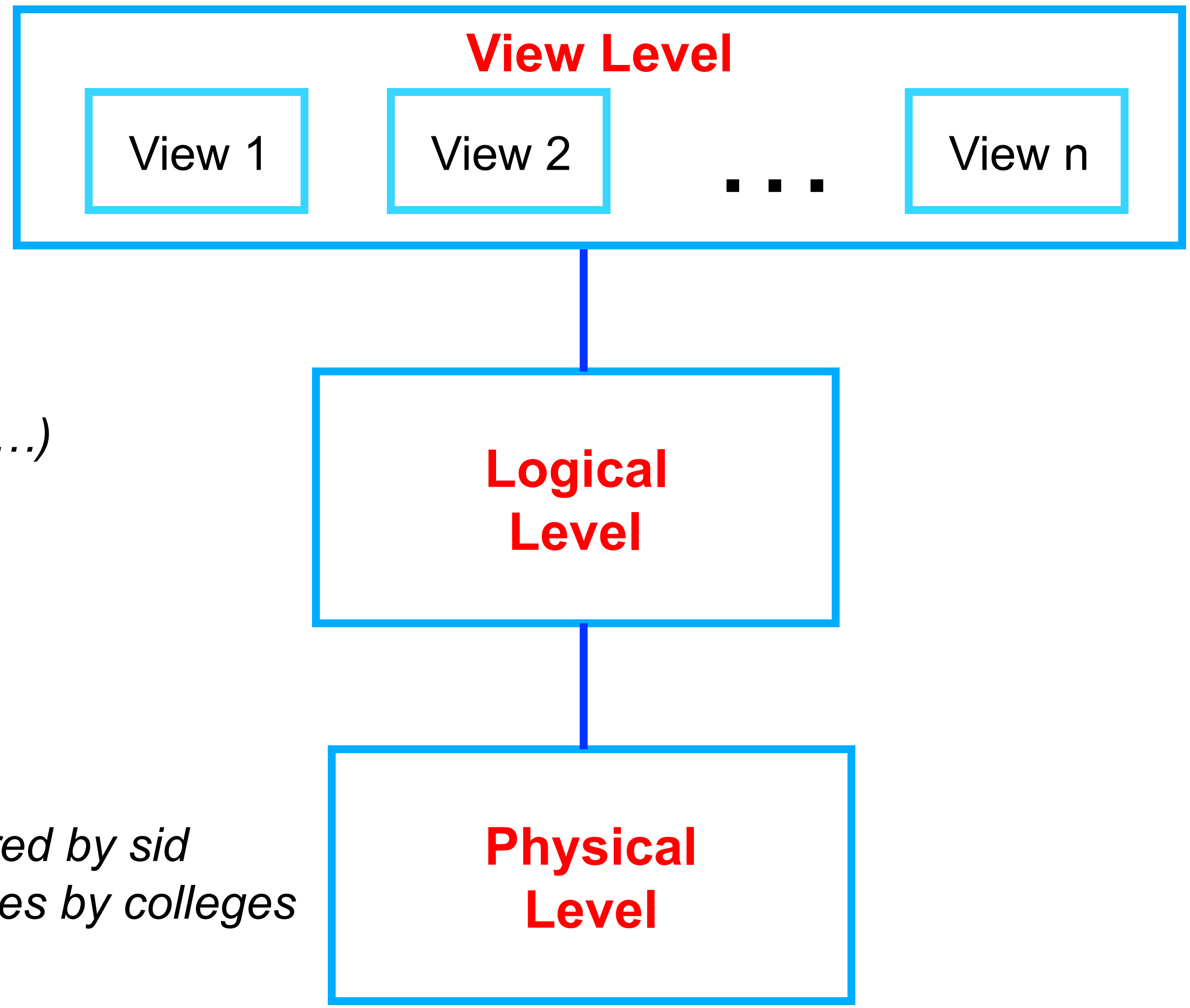


Data Abstractions: Example

A View Schema
course_info(#registered,…)

Logical Schema
students(sid, name, major, …)
courses(cid, name, …)
enrolled(sid, cid, …)

Physical Schema
all students in one file ordered by sid
courses split into multiple files by colleges



Current Industry Outlook

- Relational DBMSs
  - Oracle, IBM DB2, Microsoft SQL Server, Sybase
- Open source alternatives
  - MySQL, PostgreSQL, SQLite (primarily embedded), Apache Derby, BerkeleyDB (mainly a storage engine – no SQL), neo4j (graph data) …
- Data Warehousing Solutions
  - Geared towards very large volumes of data and on analyzing them
  - Long list: Teradata, Oracle Exadata, Netezza (based on FPGAs), Aster Data (founded 2005), Vertica (column-based), Kickfire, Xtremedata (released 2009), Sybase IQ, Greenplum (eBay, Fox Networks use them)
  - Usually sell package/services and charge per TB of managed data
  - Many (especially recent ones) start with MySQL or PostgreSQL and make them parallel/faster etc..

Web Scale Data Management, Analysis

- Ongoing debate/issue
  - Cloud computing seems to eschew DBMSs in favor of homegrown solutions
  - E.g. Google, Facebook, Amazon etc…
- MapReduce: A paradigm for large-scale data analysis
  - Hadoop: An open source implementation
  - Apache Spark: a better open source implementation
- Why ?
  - DBMSs can't scale to the needs, not fault-tolerant enough
    - These apps don't need things like transactions, that complicate DBMSs (???)
  - Mapreduce favors Unix-style programming, doesn't require SQL
    - Try writing SVMs or decision trees in SQL
  - Cost
    - Companies like Teradata may charge $100,000 per TB of data managed

Current Industry Outlook

- Bigtable-like
  - Called “key-value stores”
  - Think highly distributed hash tables
  - Allow some transactional capabilities – still evolving area
  - Apache Cassandra (Facebook), Hbase (Apache), and many many others
- Document Databases (MongoDB, ElasticSearch)
- Graph Databases (Neo4j, OrientDB, Titan)
- Mapreduce-like
  - Hadoop (open source), Pig (@Yahoo), Dryad (@Microsoft), Spark
  - Amazon EC2 Framework
  - Not really a database – but increasing declarative SQL-like capabilities are being added (e.g. HIVE at Facebook)
- Much ongoing research in industry and academia

What we will cover…

- We will mainly discuss structured data
  - That can be represented in tabular forms (called Relational data)
  - We will spend some time on XML
  - We will also spend some time on Mapreduce-like stuff
- Still the biggest and most important business (?)
  - Well defined problem with really good solutions that work
    - Contrast XQuery for XML vs SQL for relational
  - Solid technological foundations
- Many of the basic techniques however are directly applicable
  - E.g. reliable data storage etc.
  - Cf. Many recent attempts to add SQL-like capabilities, transactions to Mapreduce and related technologies
    - E.g., Spark DataFrames

What we will cover…

- representing information
  - data modeling
  - semantic constraints
- languages and systems for querying data
  - complex queries & query semantics
  - over massive data sets
- concurrency control for data manipulation
  - ensuring transactional semantics
- reliable data storage
  - maintain data semantics even if you pull the plug
  - fault tolerance

What we will cover…

- representing information
  - data modeling: *relational models, E/R models, XML/JSON*
  - semantic constraints: *integrity constraints, triggers*
- languages and systems for querying data
  - complex queries & query semantics*: SQL, Spark API*
  - over massive data sets*: indexes, query processing, optimization, parallelization/cluster processing, streaming, cluster/cloud computing*
- concurrency control for data manipulation
  - ensuring transactional semantics: *ACID properties, distributed consistency*
- reliable data storage
  - maintain data semantics even if you pull the plug: *durability*
  - fault tolerance: *RAID*

Summary

- Why study databases ?
  - Shift from *computation* to *information*
    - Always true in *corporate* domains
    - Increasing true for *personal* and *scientific* domains
  - Need has exploded in recent years
    - Data is growing at a very fast rate
  - Solving the data management problems is going to be a key
- Database Management Systems provide
  - Data abstraction: Key in evolving systems
  - Guarantees about data integrity
    - In presence of concurrent access, failures…
  - Speed !!

Computing Tools for Next Few Weeks

- git: version control system
- VirtualBox: virtualization software
- Vagrant: make it super-easy to use VirtualBox
- PostgreSQL
- Python and Jupyter Notebooks
- Instabase (optional)

Relational Model and SQL: Outline

- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

Context

- Data Models
  - Conceptual representation of the data
- Data Retrieval
  - How to ask questions of the database
  - How to answer those questions
- Data Storage
  - How/where to store data, how to access it
- Data Integrity
  - Manage crashes, concurrency
  - Manage semantic inconsistencies

Relational Data Model

Introduced by Ted Codd (late 60's – early 70's)

- *Before = "Network Data Model" (Cobol as DDL, DML)*
- *Very contentious: Database Wars (Charlie Bachman vs. Ted Codd)*

Relational data model contributes:

1. *Separation of logical, physical data models (data independence)*
2. *Declarative query languages*
3. *Formal semantics*
4. *Query optimization (key to commercial success)*

1st prototypes:

- *Ingres → CA*
- *Postgres → Illustra → Informix → IBM*
- *System R → Oracle, DB2*

Key Abstraction: Relation

Account =

bname	acct_no	balance
Downtown	A-101	500
Brighton	A-201	900
Brighton	A-217	500

Terms:

- Tables (aka: Relations)

Why called Relations?

*Closely correspond to mathematical concept of a **relation***

Relations

Account =

bname	acct_no	balance
Downtown	A-101	500
Brighton	A-201	900
Brighton	A-217	500

Considered equivalent to…

{ (Downtown, A-101, 500),
(Brighton, A-201, 900),
(Brighton, A-217, 500) }

Relational database semantics defined in terms of mathematical relations

Relations

Account =

bname	acct_no	balance
Downtown	A-101	500
Brighton	A-201	900
Brighton	A-217	500

Considered equivalent to…

{ (Downtown, A-101, 500),
(Brighton, A-201, 900),
(Brighton, A-217, 500) }

Terms:

- Tables (aka: Relations)
- Rows (aka: tuples)
- Columns (aka: attributes)
- Schema (e.g.: Acct_Schema = (bname, acct_no, balance))

Definitions

Relation Schema (or Schema)

A list of attributes and their domains

E.g. **account**(account-number, branch-name, balance)

Programming language equivalent: A variable (e.g. x)

Relation Instance

A particular instantiation of a relation with actual values

Will change with time

bname	acct_no	balance
Downtown	A-101	500
Brighton	A-201	900
Brighton	A-217	500

Programming language equivalent: Value of a variable

Definitions

Domains of an attribute/column

The set of permitted values

e.g., bname must be String, balance must be a positive real number

We typically assume domains are **atomic,** i.e., the values are treated as indivisible (specifically: you can't store lists or arrays in them)

Null value

A special value used if the value of an attribute for a row is:

- unknown (e.g., don't know address of a customer)
- inapplicable (e.g., "spouse name" attribute for a customer)
- withheld/hidden

Different interpretations all captured by a single concept – leads to major headaches and problems

Tables in a University Database

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(Id, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

Keys

- Let $K \subseteq R$
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of any possible relation r(R)
  - *Example: {ID} and {ID,name} are both superkeys of instructor.*
- Superkey K is **a candidate key** if K is minimal (i.e., no subset of it is a superkey)
  - *Example: {ID} is a candidate key for Instructor*
- One of the candidate keys is selected to be the **primary key**
  - Typically one that is small and immutable (doesn't change often)
- Primary key typically highlighted (e.g., underlined)

Tables in a University Database

classroom(building, room_number, capacity)

department(dept_name, building, budget)

course(course_id, title, dept_name, credits)

instructor(ID, name, dept_name, salary)

Tables in a University Database

takes(ID, course_id, sec_id, semester, year, grade)

What about ID, course_id?

No. May repeat:

(“1011049”, “CMSC424”, “101”, “Spring”, 2014, D)

(“1011049”, “CMSC424”, “102”, “Fall”, 2015, null)

What about ID, course_id, sec_id?

May repeat:

(“1011049”, “CMSC424”, “101”, “Spring”, 2014, D)

(“1011049”, “CMSC424”, “101”, “Fall”, 2015, null)

What about ID, course_id, sec_id, semester?

Still no: (“1011049”, “CMSC424”, “101”, “Spring”, 2014, D)

(“1011049”, “CMSC424”, “101”, “Spring”, 2015, null)

Tables in a University Database

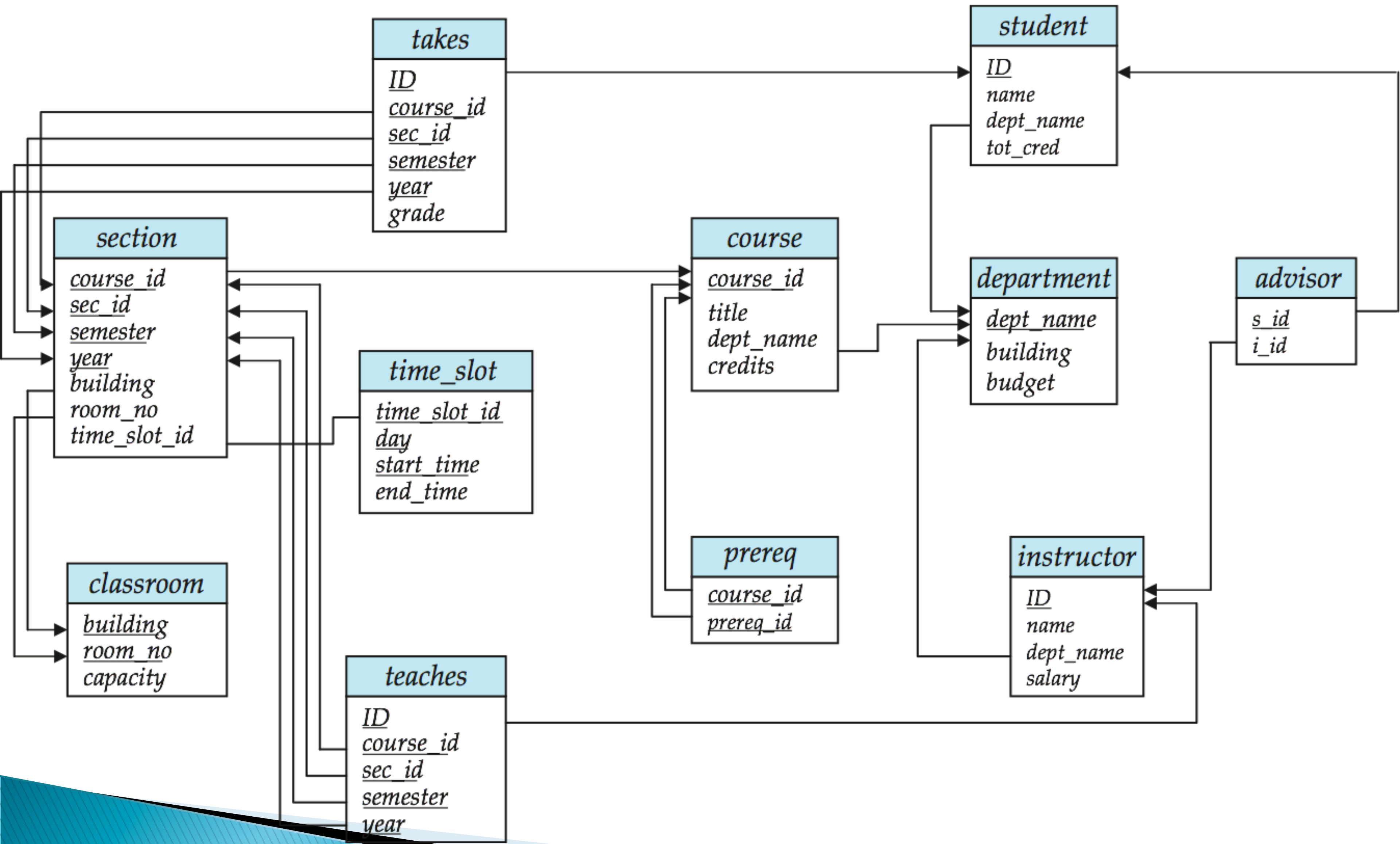
classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building,
room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

Keys

- **Foreign key:** Primary key of a relation that appears in another relation
  - {ID} from *student* appears in *takes, advisor*
  - *student* called ***referenced*** relation
  - *takes* is the ***referencing*** relation
  - Typically shown by an arrow from referencing to referenced

- **Foreign key constraint**: the tuple corresponding to that primary key must exist
  - Imagine:
    - Tuple: ('student101', 'CMSC424') in *takes*
    - But no tuple corresponding to 'student101' in *student*
  - Also called ***referential integrity constraint***

Schema Diagram for University Database



Schema Diagram for the Banking Enterprise