Data Models

- **Data Model:**
  - A set of concepts to describe the *structure* of a database, the *operations* for manipulating these structures, and certain *constraints* that the database should obey.

- **Data Model Structure and Constraints:**
  - Constructs are used to define the database structure
  - Constructs typically include *elements* (and their *data types*) as well as groups of elements (e.g. *entity, record, table*), and *relationships* among such groups
  - Constraints specify some restrictions on valid data; these constraints must be enforced at all times

Categories of Data Models

- **Conceptual (high-level, semantic) data models:**
  - Provide concepts that are close to the way many users perceive data.
  - (Also called *entity-based* or *object-based* data models.)

- **Physical (low-level, internal) data models:**
  - Provide concepts that describe details of how data is stored in the computer. These are usually specified in an ad-hoc manner through DBMS design and administration manuals

- **Implementation (representational) data models:**
  - Provide concepts that fall between the above two, used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).

- **Self-Describing Data Models:**
  - Combine the description of data with the data values. Examples include XML, key-value stores and some NOSQL systems.
Conceptual (ER) Design for mini-banner:

“Who’s taking what, and what grade do they expect?”

```
<table>
<thead>
<tr>
<th></th>
<th>fId</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROFESSOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>fid</td>
<td>name</td>
</tr>
<tr>
<td>Teaches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STUDENT</td>
<td>sid</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>exp-grade</td>
<td></td>
</tr>
<tr>
<td>COURSE</td>
<td>cid</td>
<td>subj</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>semester</td>
</tr>
</tbody>
</table>
```

Schemas versus Instances

- **Database Schema**:
  - The *description* of a database.
  - Includes descriptions of the database structure, data types, and the constraints on the database.
- **Schema Diagram**:
  - An *illustrative* display of (most aspects of) a database schema.
- **Schema Construct**:
  - A *component* of the schema or an object within the schema, e.g., STUDENT, COURSE.

Database State/Instance

- **Database State (also called instance)**:
  - The actual data (content) stored in a database at a *particular moment in time*. This includes the collection of all the data in the database.
  - **Initial Database State**:
    - Refers to the database state when it is initially loaded into the system.
  - **Valid State**:
    - A state that satisfies the structure and constraints of the database.

Database Schema vs. Database State

- **Distinction**
  - The *database schema* changes very infrequently.
  - The *database state* changes every time the database is updated.
Example of a Database Schema

**STUDENT**
- Name
- Student_number
- Class
- Major

**COURSE**
- Course_name
- Course_number
- Credit_hours
- Department

**PREREQUISITE**
- Course_number
- Prerequisite_number

**SECTION**
- Section_identifier
- Course_number
- Semester
- Year
- Instructor

**GRADE_REPORT**
- Student_number
- Section_identifier
- Grade

---

**Example Instance – Schema**

**STUDENT**
- sid
- name

**COURSE**
- sid
- exp-grade
- cid

**PROFESSOR**
- fid
- name

---

**Example of a database state**

**COURSE**
- Course_name
- Course_number
- Credit_hours
- Department

**SECTION**
- Section_identifier
- Course_number
- Semester
- Year
- Instructor

**GRADE_REPORT**
- Student_number
- Section_identifier
- Grade

---

**Relational Model Concepts**

- Our focus now: relational schema – set of tables
- Can have other kinds of schemas – XML, object, ...

- The relational Model of Data is based on the concept of a *Relation*
  - The strength of the relational approach to data management comes from the *formal foundation provided by the theory of relations*

- We start with review of the essentials of the *formal relational model*
  - In practice, there is a *standard model* based on SQL

- Note: There are several important differences between the *formal model* and the *practical model*
Relational Model Concepts

- A Relation is a mathematical concept based on the ideas of sets
- The model was first proposed by Dr. E.F. Codd of IBM Research in 1970 in the following paper:
  - "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970
    - Describes the data minimally and mathematically
    - A relation describes an association between data items – tuples with attributes
    - Uses standard mathematical (logical) operations over the data – relational algebra or relational calculus
- The above paper caused a major revolution in the field of database management and earned Dr. Codd the coveted ACM Turing Award

Relational Model

- Vendors: IBM, Microsoft, Oracle, MySQL etc.
- Shift from networked "records" of data to a set of tables
  - Why is this a good thing for SW developers?
    - Row corresponds to relationship among values
    - Table corresponds to a relation
      - mathematical definition of relation over K sets

Why Did It Take So Many Years to Implement Relational Databases?

- Codd’s original work: 1969-70
- Earliest relational database research: ~1976
- Commercial Relational DBMSs: ~mid 1980s
- Why the gap? Top 10 reasons…
  1. “You could do the same thing in other ways”
  2. “Nobody wants to write math formulas”
  3. “Why would I turn my data into tables?”
  4. “It won’t perform well”
  5. …
- *What do you think?*

Relational Model Definitions

- A relation is a table with columns and rows.
- An attribute is a named column of a relation.
- A tuple is a row of a relation.
- A domain is a set of allowable values for one or more attributes.
- The degree of a relation is the number of attributes it contains.
- The cardinality of a relation is the number of tuples it contains.
- A relational database is a collection of normalized relations with distinct relation names.
Example of a Relation

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Home_phone</th>
<th>Address</th>
<th>Office_phone</th>
<th>Age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benjamin May</td>
<td>M</td>
<td>372-1818</td>
<td>2618 Bluebonnet Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.21</td>
</tr>
<tr>
<td>Chung-Chi Kim</td>
<td>F</td>
<td>375-4409</td>
<td>125 Kirby Road</td>
<td>138-1245</td>
<td>18</td>
<td>2.89</td>
</tr>
<tr>
<td>Dick Davidson</td>
<td>M</td>
<td>NULL</td>
<td>3452 Elgin Road</td>
<td>749-1253</td>
<td>25</td>
<td>3.00</td>
</tr>
<tr>
<td>Rohan Pandit</td>
<td>M</td>
<td>376-9821</td>
<td>265 Lark Lane</td>
<td>749-6492</td>
<td>28</td>
<td>3.00</td>
</tr>
<tr>
<td>Barbara Benson</td>
<td>F</td>
<td>639-9401</td>
<td>7384 Fontana Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.25</td>
</tr>
</tbody>
</table>

Degree = 7, Cardinality = 5

Relation Schemas and Instances

- A **relation schema** is a definition of a single relation.
  - The relation schema is the intension of the relation.
- A **relational database schema** is a set of relation schemas (modeling a particular domain).

- A **relation instance** denoted r(R) over a relation schema \( R(A_1, A_2, \ldots, A_n) \) is a subset of the Cartesian product of the domains of all attributes in the relation schema \( r(R) \subseteq \text{dom}(A_1) \times \text{dom}(A_2) \times \cdots \times \text{dom}(A_n) \).
  - i.e., a set of n-tuples \( <d_1, d_2, \ldots, d_n> \) where each \( d_i \) is an element of \( \text{dom}(A_i) \) or is null.
  - A value of null represents a missing or unknown value.

Relational Model: Formal Definition

- Formally, a table is a relation over K sets (domains)
  - \( R \subseteq A_1 \times A_2 \ldots \times A_K \)
    - Subset of the cartesian product of the K domains
  - Tuple = \( (t_1, t_2, \ldots, t_k) \), where \( t_i \) is an element from domain/set \( D_i \)

- A database is a collection of relations
- Theoretically: a relation is a set of tuples; no tuple can occur more than once
  - Real systems may allow duplicates for efficiency or other reasons – we’ll ignore this for now

Definition Summary

<table>
<thead>
<tr>
<th>Informal Terms</th>
<th>Formal Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Relation</td>
</tr>
<tr>
<td>Column Header</td>
<td>Attribute</td>
</tr>
<tr>
<td>All possible Column</td>
<td>Domain</td>
</tr>
<tr>
<td>Values</td>
<td></td>
</tr>
<tr>
<td>Row</td>
<td>Tuple</td>
</tr>
<tr>
<td>Table Definition</td>
<td>Schema of a Relation</td>
</tr>
<tr>
<td>Populated Table</td>
<td>State of the Relation</td>
</tr>
</tbody>
</table>
Properties of relations

- Each relation name is unique
  - No two relations have the same name
- Each cell of the relation (value of a domain) contains exactly one atomic (single) value
- Each attribute of a relation has a distinct name
- Values of an attribute are all from the same domain
- Each tuple is distinct. There are no duplicate tuples
  - Properties of a relation; in SQL can be bags (allow duplicates)
- Order of attributes is not really important
  - Note difference from mathematical def of relations
  - Reason: attribute names represent domain and can be reordered
- Order of tuples if not important

Properties and constraints of the data

- Data represented as a relation/table
- Schema specifies the attributes/columns and their type
- Anything other properties we need to define to capture the "application"
- Need to capture the 'business' rules:
  - How do we uniquely identify a student?
  - Can their letter grade be any alphabet?
  - Can a student take a course that is not in the course schedule/bulletin?
  - ….
- Concept of Constraints

CONSTRAINTS

Constraints determine which values are permissible and which are not in the database. Three main types:

- **Inherent or Implicit Constraints**: These are based on the data model itself. (E.g., relational model does not allow a list as a value for any attribute)
- **Schema-based or Explicit Constraints – Integrity Constraints**: They are expressed in the schema by using the facilities provided by the model.
- **Application based or semantic constraints**: These are beyond the expressive power of the model and must be specified and enforced by the application programs.

Integrity Constraints (ICs)

- **IC**: condition that must be true for any instance of the database; e.g., *domain constraints*
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- Why is this useful
  - If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!
- Think of the constraints as the business rules derived from the application
Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
  - Need to carefully analyze the application before reaching a conclusion on the Integrity Constraints!
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know name is not a key, but the assertion that sid is a key is given to us.
- Key and foreign key ICs are the most common; more general ICs supported too.
  - Shall return to these after we cover DML aspect of SQL

Relational Integrity

- Integrity rules are used to insure the data is accurate.
- Constraints are rules or restrictions that apply to the database and limit the data values it may store.
- Types of constraints:
  - Domain constraint - Every value for an attribute must be an element of the attribute's domain or be null.
    null represents a value that is currently unknown or not applicable.
    null is not the same as zero or an empty string.
  - Entity integrity constraint - In a base relation, no attribute of a primary key can be null.
  - Key constraint – every relation must have a key
  - Referential integrity constraint - If a foreign key exists in a relation, then the foreign key value must match a primary key value of a tuple in the referenced relation or be null.

Key Constraints

- Superkey of R:
  - Is a set of attributes SK of R with the following condition:
    No two tuples in any valid relation state r(R) will have the same value for SK
    That is, for any distinct tuples t1 and t2 in r(R), t1[SK] ≠ t2[SK]
    This condition must hold in any valid state r(R)
- Key of R:
  - A "minimal" superkey
  - That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)
  - A Key is a Superkey but not vice versa

Primary Key Constraints

- Every relation must have a key
- A set of fields is a key for a relation if:
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
    - Part 2 false? A superkey.
    - If there’s >1 key for a relation, one of the keys is chosen (by DBA) to be the primary key.
- E.g., what is a key for Students relation?
  - sid is a key for Students. (What about name?) The set {sid, gpa} is a superkey.
Key Constraints (continued)
- Example: Consider the CAR relation schema:
  - CAR(State, Reg#, SerialNo, Make, Model, Year)
  - CAR has two keys:
    - Key1 = (State, Reg#)
    - Key2 = (SerialNo)
  - Both are also superkeys of CAR
  - {SerialNo, Make} is a superkey but not a key.
- In general:
  - Any key is a superkey (but not vice versa)
  - Any set of attributes that includes a key is a superkey
  - A minimal superkey is also a key
  - Can have many candidate keys, one of them chosen as primary key

Key Constraints (continued)
- If a relation has several candidate keys, one is chosen arbitrarily to be the primary key.
  - The primary key attributes are underlined.
- Example: Consider the CAR relation schema:
  - CAR(State, Reg#, SerialNo, Make, Model, Year)
  - We chose SerialNo as the primary key
  - The primary key value is used to uniquely identify each tuple in a relation
  - Provides the tuple identity
  - Also used to reference the tuple from another tuple
  - General rule: Choose as primary key the smallest of the candidate keys (in terms of size)
  - Not always applicable – choice is sometimes subjective

Referential Integrity
- A constraint involving two relations
  - The previous constraints involve a single relation.
  - Used to specify a relationship among tuples in two relations:
    - The referencing relation and the referenced relation.
- Examples:
  - Student enrolled in a course
  - Employee working on a project: in Works_On table

Foreign Keys, Referential Integrity
- **Foreign key**: Set of fields in one relation that is used to ‘refer’ to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a ‘logical pointer’.
- In Enrolled table – sid is a student, what can we say about the students table?
  - Enrolled(sid: string, cid: string, grade: string)
- sid is a foreign key referring to Students:
  - If all foreign key constraints are enforced, referential integrity is achieved, i.e., no dangling references.
Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.

<table>
<thead>
<tr>
<th>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sid</td>
</tr>
<tr>
<td>53666</td>
<td>53666</td>
</tr>
<tr>
<td>53688</td>
<td>53688</td>
</tr>
<tr>
<td>53650</td>
<td>53650</td>
</tr>
<tr>
<td>53666</td>
<td></td>
</tr>
</tbody>
</table>

Referential Integrity (or foreign key) Constraint – more general definition

- Statement of the constraint
  - The value in the foreign key column (or columns) FK of the referencing relation R1 can be either:
    1. a value of an existing primary key value of a corresponding primary key PK in the referenced relation R2, or
    2. a null.
- In case (2), the FK in R1 should not be a part of its own primary key.

Other Types of Constraints

- Semantic Integrity Constraints:
  - based on application semantics and cannot be expressed by the model per se
  - Example: “the max. no. of hours per employee for all projects he or she works on is 40 hrs per week”; Grade cannot be any alphabet; …
  - A constraint specification language may have to be used to express these
- SQL-99 allows CREATE TRIGGER and CREATE ASSERTION to express some of these semantic constraints
- Keys, Permissibility of Null values, Candidate Keys (Unique in SQL), Foreign Keys, Referential Integrity etc. are expressed by the CREATE TABLE statement in SQL.

When do Integrity Constraints get triggered...
Relational Database Schema

- Relational Database Schema:
  - A set $S$ of relation schemas that belong to the same database.
  - $S$ is the name of the whole database schema
  - $S = \{R_1, R_2, ..., R_n\}$ and a set $IC$ of integrity constraints.
  - $R_1, R_2, ..., R_n$ are the names of the individual relation schemas within the database $S$

- Following slide shows a COMPANY database schema with 6 relation schemas

Populated database state

- Whenever the database is changed, a new state arises
- Basic operations for changing the database:
  - INSERT a new tuple in a relation
  - DELETE an existing tuple from a relation
  - MODIFY an attribute of an existing tuple
- Next slide shows an example state for the COMPANY database schema
- The update operations must keep the database in a consistent state – i.e. all instances must satisfy integrity constraints
- Updates may propagate to cause other updates automatically. This may be necessary to maintain integrity constraints.
**Update Operations on Relations**

- In case of integrity violation, several actions can be taken:
  - Cancel the operation that causes the violation (RESTRICT or REJECT option)
  - Perform the operation but inform the user of the violation
  - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  - Execute a user-specified error-correction routine

**Possible violations for each operation**

- **INSERT** may violate any of the constraints:
  - Domain constraint:
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain
  - Key constraint:
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation
  - Referential integrity:
    - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
  - Entity integrity:
    - if the primary key value is null in the new tuple

- **DELETE** may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
    - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL (see Chapter 6 for more details)
      - RESTRICT option: reject the deletion
      - CASCADE option: propagate the new primary key value into the foreign keys of the referencing tuples
      - SET NULL option: set the foreign keys of the referencing tuples to NULL
  - One of the above options must be specified during database design for each foreign key constraint

- **UPDATE** may violate domain constraint and NOT NULL constraint on an attribute being modified
  - Any of the other constraints may also be violated, depending on the attribute being updated:
    - Updating the primary key (PK):
      - Similar to a DELETE followed by an INSERT
      - Need to specify similar options to DELETE
    - Updating a foreign key (FK):
      - May violate referential integrity
    - Updating an ordinary attribute (neither PK nor FK):
      - Can only violate domain constraints

- **UPDATE** may violate domain constraint and NOT NULL constraint on an attribute being modified
  - Any of the other constraints may also be violated, depending on the attribute being updated:
    - Updating the primary key (PK):
      - Similar to a DELETE followed by an INSERT
      - Need to specify similar options to DELETE
    - Updating a foreign key (FK):
      - May violate referential integrity
    - Updating an ordinary attribute (neither PK nor FK):
      - Can only violate domain constraints
Relational Query Languages

- A major strength of the relational model: supports simple, powerful querying of data.
  Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - The key: precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

Relational Query Languages

- Query languages:
  - Allow specification of schemas and constraints
  - Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- Query Languages != programming languages!
  - QLs not expected to be “Turing complete”.
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

Formal Query Languages

- Formal query languages are defined as mathematical operators over the set
  - What is the advantage of a formal language?
  - Relational algebra, Relational calculus are examples
- Procedural vs Non-procedural languages
  - Procedural: what data to fetch from DB and how/where to get the data
  - Non-procedural: what data to fetch from DB
    System/DBMS needs to figure out the “how”
  - Can have a mix in practice
    - Relational algebra: procedural language
    - Relational calculus: non-procedural (declarative)

SQL: Structured Query Language

- The standard language for relational data
  - Invented by folks at IBM, esp. Don Chamberlin
  - Actually not a great language…
  - Beat a more elegant competing standard, QUEL, from Berkeley
- Separated into a DML & DDL
- SQL DML component based on relational algebra & calculus
SQL

- components
  - Data definition (DDL) – to define schema/tables
    - Define Schema
    - Define Constraints
  - Manipulation/query (DML) – for queries
  - Transaction control – to specify a transaction
  - Index – to specify storage and indexing schemes
  - Authorization- for access control/security

  We will cover the DDL and query part of SQL first
  Shall return to the other components after we cover those topics

Summary....and next

- Relational model
- Schema definition
- Constraints

Next:
- SQL DDL to define schema and constraints
- Formal Query Languages:
  - Relational Algebra
  - Relational tuple calculus
- SQL
- In Lab:
  - Get started with your account (This is homework)
  - Intro to PHP
- Writing Homework 1 posted