

# The Relational Data Model

Davood Rafiei

Original material Copyright 2001-2019  
(some material from textbooks and other instructors)



# Data Model

- Need a model for describing
  - The structure of data and constraints
  - Operations on data
- Describe data at some abstraction layer (schema)
  - External
  - Conceptual
  - Physical



# Physical Level

- Data description: e.g.
  - Student data is stored in tracks 4 to 15 of Cylinder 3.
  - Student data is sorted on student id.
  - Course data is stored in an ASCII file named courses.dat
  - Records are separated by new-line characters.
  - Fields are separated by commas.
- Data manipulation: e.g.
  - Read all data in track 2 of cylinder 4.
  - Find John's record in the data just read.
  - Update John's record.
  - Write back the data in track 2 of cylinder 4.



# Physical Level (Cont)

- **Problems of working with data:** Routines hard-coded to deal with physical representation.
  - Difficult to change the physical representation.
  - Application code becomes complex since it must deal with details.
  - Rapid implementation of new features impossible.



# Conceptual Level

- Hides details.
  - In the relational model, the conceptual schema presents data as a set of tables.
- Mapping from conceptual to physical schema done by DBMS.
- Physical schema can be changed without changing applications.
  - Referred to as *physical data independence*.



# Relational Databases

- **Basic idea:**
  - Organize data as a set of tables.
  - View each table as a set of rows.
- **Advantages**
  - simple
  - solid mathematical foundation
    - ✓ set theory
  - powerful query languages
  - efficient query optimizers



# Definition

- *Relational database*: a set of relations (tables)
- *Relation*: consists of
  - *Instance* : *table content*, with rows and columns.
    - ✓ #Rows = *cardinality*, #fields = *degree / arity*.
  - *Schema* : *table structure*, with name and type of columns.
    - ✓ E.G. Students(*sid*: char(8), *name*: char(16), *login*: char(8), *age*: int, *gpa*: float).
- More formally, a relation is a *set* of rows (or *tuples*)
  - What does this imply for each tuple?



# Example: Students Relation

| sid   | name | login       | age | gpa |
|-------|------|-------------|-----|-----|
| 10310 | Mary | mary@cs     | 19  | 3.6 |
| 10400 | Bob  | bob@physics | 20  | 3.2 |
| 11001 | Bob  | bob@math    | 18  | 3.8 |

- Cardinality = 3, degree = 5, all rows distinct
- **Domain**: the set of values from which the values of an attribute are drawn.
  - e.g. name : char(16), age : {1,...,100}





# Querying Relations

- Simple and powerful *querying* of data.
  - Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - Precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.



# Example: Querying Relations in SQL

- To find all 18 year old students, we can write:

```
SELECT *  
FROM Students S  
WHERE S.age=18;
```

| sid   | name | login       | age | gpa |
|-------|------|-------------|-----|-----|
| 10310 | Mary | mary@cs     | 19  | 3.6 |
| 10400 | Bob  | bob@physics | 20  | 3.2 |
| 11001 | Bob  | bob@math    | 18  | 3.8 |

- To find just names and logins, replace the first line with:

```
SELECT S.name, S.login
```



# Example: Querying Multiple Relations

- What does the following query do?

```
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid=E.sid AND E.grade='A';
```

Students

| sid   | name | login       | age | gpa |
|-------|------|-------------|-----|-----|
| 10310 | Mary | mary@cs     | 19  | 3.6 |
| 10400 | Bob  | bob@physics | 20  | 3.2 |
| 11001 | Bob  | bob@math    | 18  | 3.8 |

Enrolled

| sid   | cid       | grade |
|-------|-----------|-------|
| 10310 | CMPUT 201 | C     |
| 10310 | CMPUT 291 | B     |
| 10400 | BIO 101   | A     |
| 11001 | Math 101  | B     |

result:

| S.name | E.cid   |
|--------|---------|
| Bob    | BIO 101 |



# Creating Relations in SQL

- Two tables (as examples)
- The type (**domain**) of each field is specified by user/programmer, and enforced by the DBMS (whenever tuples are added or modified).

```
CREATE TABLE Students  
(sid: CHAR(5),  
name: CHAR(10),  
login: CHAR(15),  
age: INT,  
gpa: FLOAT);
```

```
CREATE TABLE Enrolled  
(sid: CHAR(5),  
cid: CHAR(10),  
grade: CHAR(2));
```



# Adding and Deleting Tuples

- Insert a single tuple

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (11010, 'Andrew', 'andrew@cs', 18, 3.3);
```

- Delete all tuples that satisfy a condition (e.g., name = Bob):

```
DELETE
FROM Students
WHERE name = 'Bob';
```

☞ *Will see more powerful versions of these commands later!*



# Integrity Constraints (ICs)

- Conditions that hold for *any* instance of the database; e.g., *domain constraints*.
  - defined when schema is defined.
  - checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!



# Primary Key Constraints

- A set of fields is a key for a relation if it is both:
  1. unique: no two distinct tuples can have same values in all key fields, and
  2. minimal: no subset of a key is a key.
- A relation can have more than one key
  - *Candidate key*: all keys of the relation
  - *Primary key*: one defined by DBA
- *Superkey*: 1<sup>st</sup> condition holds but the 2<sup>nd</sup> may not
- E.g., *sid* is a key for Students. What about *name*? The set {*sid*, *gpa*} is a superkey.



# Primary and Candidate Keys in SQL

- A table can have many candidate keys (specified using **UNIQUE**), but only one *primary key*.
- “Given a student and a course in Enrolled1, there is a single grade.”

```
CREATE TABLE Enrolled1
(sid CHAR(8),
 cid CHAR(8),
 grade CHAR(2),
PRIMARY KEY (sid,cid) );
```
- What are the constraints in Enrolled2?

```
CREATE TABLE Enrolled2
(sid CHAR(8),
 cid CHAR(8),
 grade CHAR(2),
PRIMARY KEY (sid),
UNIQUE (cid, grade) );
```
- Used carelessly, an IC can prevent the storage of database instances that arise in practice!





# Foreign Keys, Referential Integrity

- Foreign key : Set of fields in one relation that `refers' to a tuple in another relation; a FK must correspond to a key (primary or candidate) of the other relation (like a `logical pointer').
  - E.g. *sid* in Enrolled is a foreign key referring to **Students**:
    - ✓ Enrolled(*sid*: char(8), *cid*: char(8), *grade*: char(2))
  - Referential integrity is achieved if all foreign key constraints are enforced, i.e., no dangling references.
  - Can you name a data model w/o referential integrity?



# Foreign Keys in SQL

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

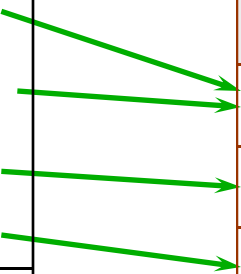
```
CREATE TABLE Enrolled  
  (sid CHAR(8), cid CHAR(8), grade CHAR(2),  
   PRIMARY KEY (sid,cid),  
   FOREIGN KEY (sid) REFERENCES Students );
```

Enrolled

| sid   | cid       | grade |
|-------|-----------|-------|
| 10310 | CMPUT 201 | C     |
| 10310 | CMPUT 291 | B     |
| 10400 | BIO 101   | A     |
| 11001 | Math 101  | B     |

Students

| sid   | name | login       | age | gpa |
|-------|------|-------------|-----|-----|
| 10310 | Mary | mary@cs     | 19  | 3.6 |
| 10400 | Bob  | bob@physics | 20  | 3.2 |
| 11001 | Bob  | bob@math    | 18  | 3.8 |



# Enforcing Referential Integrity

- *sid* in Enrolled is a foreign key that references Students.
- What if an Enrolled tuple with a non-existent student id is inserted?
  - *Reject it!*
- What if a tuple in Students is deleted?
  - Delete all Enrolled tuples that refer to it.
  - Disallow deletion of a Students tuple that is referred to.
  - Set *sid* in Enrolled tuples that refer to it to a *default sid*.  
(In SQL, also: Set *sid* in Enrolled tuples that refer to it to a special value *null*, denoting '*unknown*' or '*inapplicable*'.)
- Similar if primary key of Students tuple is updated.



# Referential Integrity in SQL/92

- SQL/92 (and SQL/99) supports all options on deletes and updates.
  - Default is **NO ACTION** (*delete/update is rejected*)
  - **CASCADE** (also delete all tuples that refer to deleted tuple)
  - **SET NULL / SET DEFAULT** (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
(sid CHAR(8),
cid CHAR(8),
grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid)
REFERENCES Students
ON DELETE CASCADE
ON UPDATE SET NULL );
```

SQLite also has **RESTRICT** which is very similar to **NO ACTION**

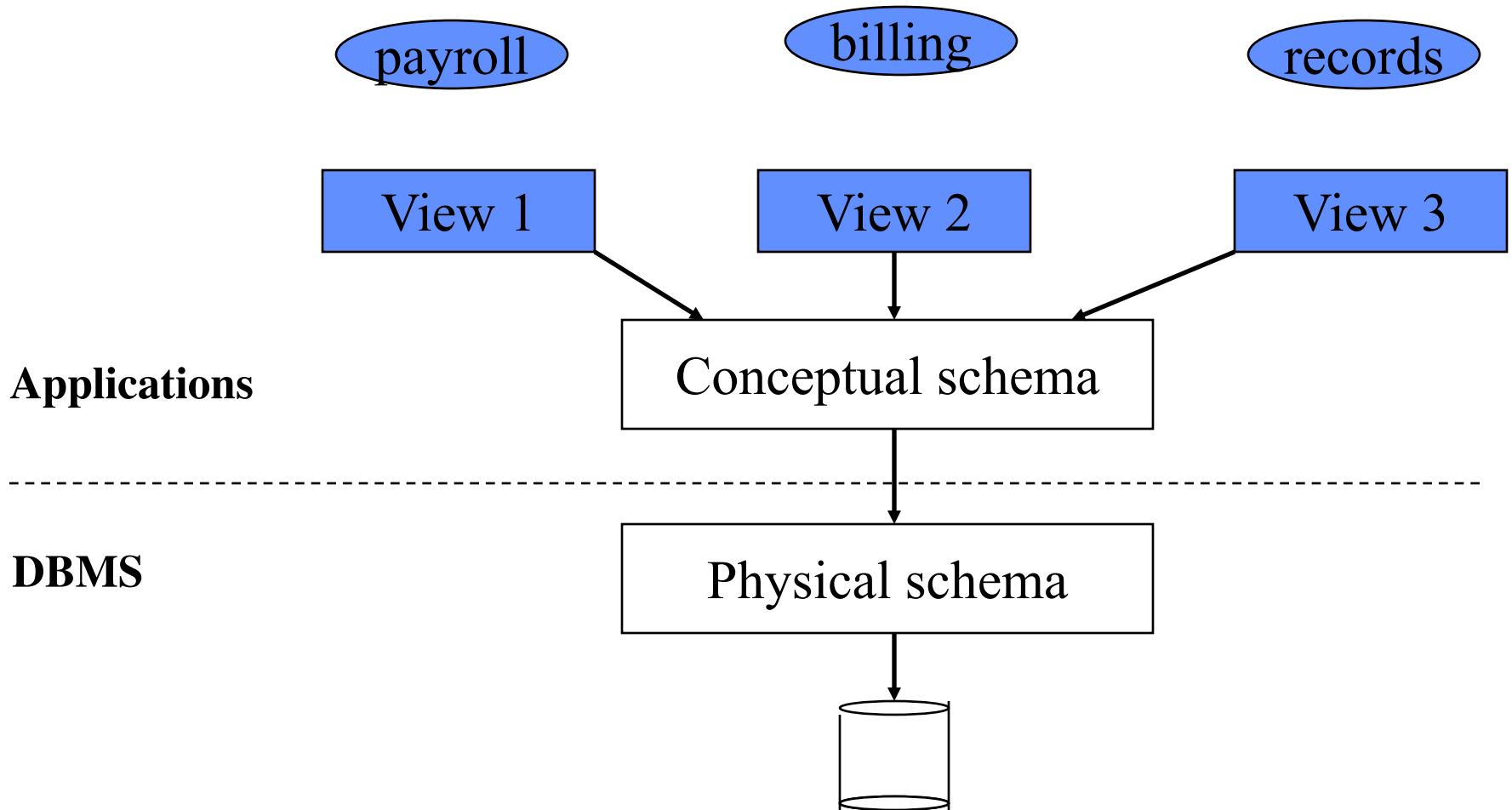


# Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
  - An IC is a statement about *all possible* instances!
  - We cannot infer that an IC is true by looking at an instance.
  - 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.



# Levels of Abstraction



# External Level

- Applications can access data through some views
  - Different views of data for different categories of users
  - A view is computed (from data in the conceptual level)
- Mapping from external to conceptual schema is done by DBMS.
- Conceptual schema can be changed without changing applications:
  - Referred to as *logical data independence*.



# Views

- A view is just a relation, but we store a *definition*, rather than a set of tuples.

```
CREATE VIEW YoungActiveStudents (name)
AS SELECT S.name
FROM Students S, Enrolled E
WHERE S.sid = E.sid and S.age<21;
```

- ❖ Views can be dropped using the **DROP VIEW** command.





# Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given YoungActiveStudents, but not Students or Enrolled, we can find young students who are enrolled, but not the *cid*'s of or *grade*'s in the courses they are enrolled in.



# Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used.
- Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
  - Two important ICs: primary and foreign keys
  - In addition, we *always* have domain constraints.
- Powerful and natural query languages exist.



# Relational Model: Summary

- Most widely used model.
  - Vendors: IBM, Informix (bought by IBM), Microsoft, Oracle, Sybase (bought by SAP), SQLite, etc.
- Old competitors:
  - hierarchical model, network model
- Recent competitors:
  - object-oriented model
    - ✓ ObjectStore, Versant, Ontos
    - ✓ A synthesis emerging: *object-relational model*
      - ✓ Informix Universal Server, O2, Oracle, DB2
  - XML graph model

