Relational data model

Iztok Savnik
FAMNIT

Slides are based on

- *Raghu Ramakrishnan, Johannes Gehrke, Database Management Systems, McGraw-Hill, 3rd ed., 2007.*
- *Slides from „Cow Book“: R.Ramakrishnan, http://pages.cs.wisc.edu/~dbbook/*

Why Study the Relational Model?

- Most widely used model
  - Relations have strong mathematical background
  - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
  - 80% of all database systems are relational
- 1990: Object-oriented movement
  - ObjectStore, ObjectDatabase++, GemStone/S, Versant, Ontos, ZODB, Wakanda, ObjectDB, …
  - Only few OO DBMS products available in 2000
- 2000: Object-relational model
  - Relations of objects mapped to flat relational model
  - Implementations: Oracle, DB2, Sybase, SQL3, etc.
  - OR model is still not used widely!

Why Study the Relational Model?

- 2000-2010: NoSQL DBMS movement
  - What happened?
    - Big data, internet info. systems, unstructured data, semi-structured data, scientific data, pictures, videos, etc.
    - Existing relational DBMS-s are not capable of scaling to manage big data
  - At the same time
    - New technology: huge RAM, SSD disks, huge hard disks, highly parallel and distributed architectures, multi-processor and multi-core architectures, shared-nothing systems, itd.
  - Result:
    - Rise of No-SQL Systems!
    - Products: Key/value stores, Document stores, Column stores, Graph DBMS, In-memory DBMS, etc.

Why Study the Relational Model?

  - Existing NoSQL systems:
    - MongoDB, CauchDB, Berkeley DB, Dynamo, Hbase, Bigtable, Hypertable, Cassandra, Sybase IQ, Vertica, ArangoDB, OrientDB, Neo4j, GraphDB, Dgraph, Virtuoso, ...
- 2010-2020: NewSQL systems
  - Technology developed is used in New relational DBMS-s!
  - Recent relational DBMS newcomers:
    - Google: Megastore (2011), Spaner (2012), F1 (2013)
    - Amazon: RDS
  - Other RDBMS vendors include new technology
    - Improved scalability and availability
    - Oracle, DB2 (IBM), Sybase, etc.

Why Study the Relational Model?

- New DBMSs presented in course:
  - ”Database systems for big data”, M.Sc. Program, CS, FAMNIT
- Relational systems have the largest market
  - ... also after »revolution«
  - After the initial excitement the NoSQL flow calmed down
  - Research area has well organized body of knowledge
  - R-DBMS features: Consistentcy, efficient optimizer, transactions, parallel execution, reliability, crash recovery, distribution, scalability and availability.

Relational Database: Definitions

- *Relational database:* a set of *relations*
- *Relation:* made up of 2 parts:
  - *Instance* : a *table*, with rows and columns. #Rows = *cardinality*, #fields = *degree / arity.*
  - *Schema* : specifies name of relation, plus name and type of each column.
    - E.G. Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real).
- Can think of a relation as a *set* of rows or *tuples* (i.e., all rows are distinct).

Example Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- Cardinality = 3, degree = 5, all rows distinct
- Do all columns in a relation instance have to be distinct?

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.

The SQL Query Language

- Developed by IBM (system R) in the 1970s
- Need for a standard since it is used by many vendors
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision)
  - SQL3 (1999, major extensions, current standard)
  - SQL3: 2003, 2006, 2008, 2011, 2017, 2019

The SQL Query Language

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

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

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

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

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

Querying Multiple Relations

- What does the following query compute?

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

Given the following instances of Enrolled and Students:

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

we get:

S.name	E.cid
Smith	Topology112

Creating Relations in SQL

- Creates the Students relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.
- As another example, the Enrolled table holds information about courses that students take.

```
CREATE TABLE Students
         (sid: CHAR(20),
          name: CHAR(20),
          login: CHAR(10),
          age: INTEGER,
          gpa: REAL)
```

```
CREATE TABLE Enrolled
         (sid: CHAR(20),
          cid: CHAR(20),
          grade: CHAR(2))
```

Destroying and Altering Relations

```
DROP TABLE  Students
```

- Destroys the relation Students. The schema information *and* the tuples are deleted.

```
ALTER TABLE  Students
        ADD COLUMN firstYear: integer
```

- The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO  Students (sid, name, login, age, gpa)
VALUES  (53688, ‘Smith’, ‘smith@ee’, 18, 3.2)
```

❖ Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = ‘Smith’
```

☛ *Powerful variants of these commands are available; more later!*

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.
- 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 :
  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., *sid* is a key for Students. (What about *name*?) The set {*sid, gpa*} is a superkey.

Primary and Candidate Keys in SQL

- Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.

- "For a given student and course, there is a single grade." vs. "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."
- Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid) )
```

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid),
   UNIQUE (cid, grade) )
```

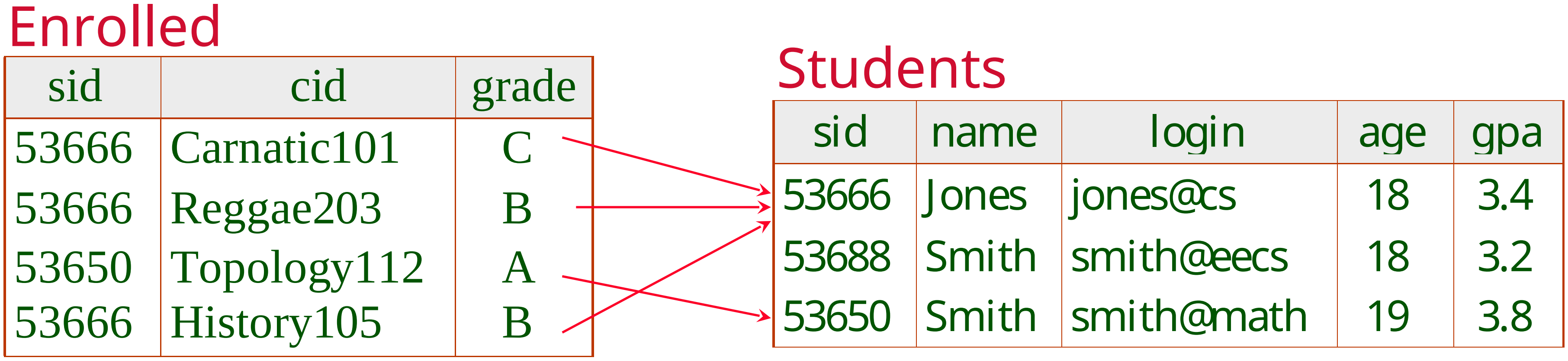
Foreign Keys, Referential Integrity

- *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'.
- E.g. *sid* is a foreign key referring to Students:
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.
  - Can you name a data model w/o referential integrity?
    - Links in HTML!

Foreign Keys in SQL

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

```
CREATE TABLE Enrolled
  (sid CHAR(20),  cid CHAR(20),  grade CHAR(2),
   PRIMARY KEY  (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students )
```



Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enforcing Referential Integrity

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a Students tuple is deleted?
  - Also 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

- SQL/92 and SQL:1999 support all 4 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(20),
    cid CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid,cid),
    FOREIGN KEY (sid)
      REFERENCES Students
            ON DELETE CASCADE
            ON UPDATE SET DEFAULT )
```

Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- 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.