Introduction to database systems: The relational model

Adaptation of slides by Prof. Manos Athanassoulis

https://midas.bu.edu/classes/CS460/

The Relational Model

Intro & SQL overview

Keys & Integrity Constraints

The Relational Model

Intro & SQL overview

Keys & Integrity Constraints

Why the Relational Model?

most widely used model
IBM, Microsoft, Oracle, etc.

”Legacy systems” in older models
e.g., IBM’s IMS

object-relational model incorporates oo concepts
IBM DB2, Oracle 11i

more recently: key-value store

Relational

tables with rows and columns

well-defined schema

data model fits data rather than functionality

deduplication

Relational Database: Definitions

relational database: a collection (set) of *relations*

each relation: made up of 2 parts

schema: name of relation, name & type of each column

Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)

instance : a *table,* with rows and columns.

#rows = *cardinality*

#fields = *degree / arity*

can think of a relation as a *set* of rows or *tuples*

(1) all rows are distinct

(2) no order among rows

Instance of Students Relation

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

cardinality = 3, arity = 5, all rows distinct

do all values in each column of a relation instance have to be distinct?

SQL - A language for Relational DBs

SQL* (a.k.a. “Sequel”), standard language

Data Definition Language (DDL)

create, modify, delete relations

specify constraints

administer users, security, etc.

Data Manipulation Language (DML)

specify *queries* to find tuples that satisfy criteria

add, modify, remove tuples

* Structured Query Language

SQL Overview

```
CREATE TABLE <name> ( <field> <domain>, … )

INSERT INTO <name> (<field names>)
     VALUES (<field values>)

DELETE FROM <name>
      WHERE <condition>

UPDATE <name>
   SET <field name> = <value>
 WHERE <condition>

SELECT <fields>
  FROM <name>
 WHERE <condition>
```

Creating Relations in SQL

type (domain) of each field is specified

also enforced whenever tuples are added or modified

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

Table Creation (continued)

Enrolled: holds information about courses students take

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

Adding and Deleting Tuples

Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
 VALUES  (‘53688’, ‘Smith’, ‘smith@cs’, 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!

The Relational Model

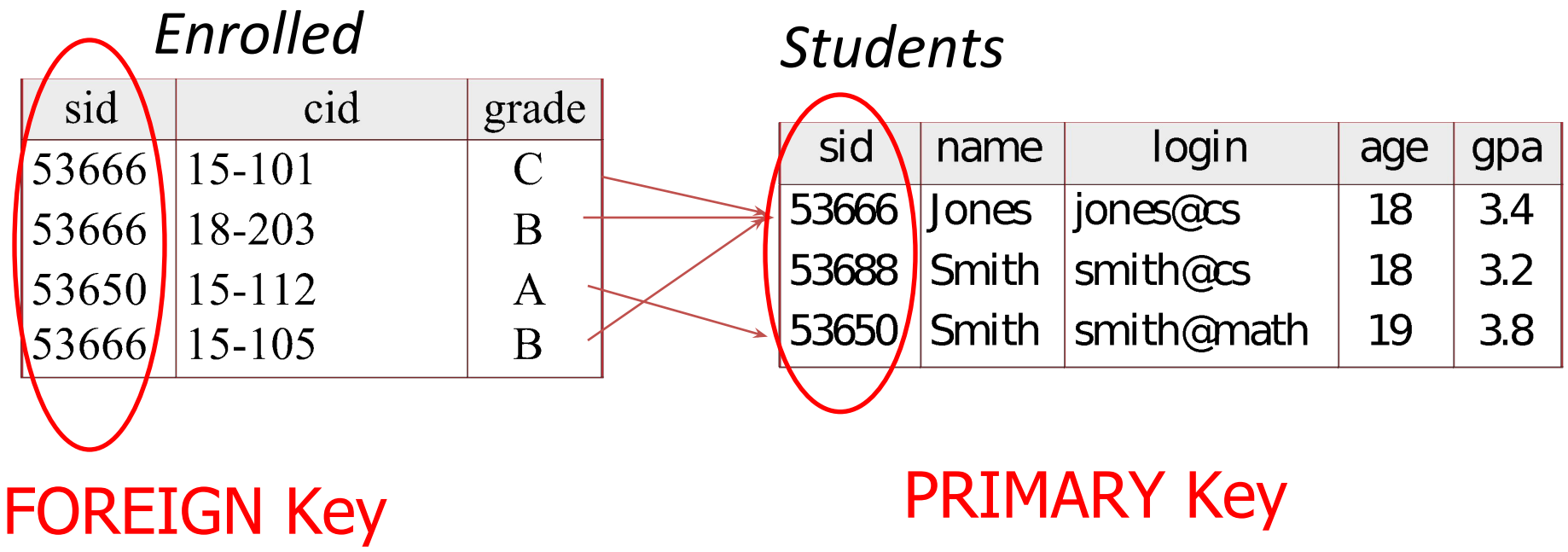
Intro & SQL overview

Keys & Integrity Constraints

Keys

keys: associate tuples in different relations

keys are one form of integrity constraint (IC)



Primary Keys

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

A set of fields is a *superkey* if:

No two distinct tuples can have same values in all key fields

Is <sid> a superkey?

What about <sid,name>?

What about <sid,name,age>?

What about <age,name>?

Primary Keys

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

A set of fields is a *superkey* if:

No two distinct tuples can have same values in all key fields

A set of fields is a *key* for a relation if :

It is a superkey

No subset of the fields is a superkey

Is <sid> a key? <sid,name>? <sid,name,age>? <age,name>?

Primary Keys

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

A set of fields is a *superkey* if:

No two distinct tuples can have same values in all key fields

A set of fields is a *key* for a relation if :

It is a superkey

No subset of the fields is a superkey

what if >1 key for a relation?

chose one as the ***primary key*** / rest called ***candidate*** keys

Primary and Candidate Keys in SQL

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

keys must be defined carefully!

"for a given student and course, there is a single grade"

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

vs.

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

"students can take only one course, and no two students in a course receive the same grade"

Foreign Keys, Referential Integrity

foreign key: set of fields in one relation that is used to "refer" to a tuple in another

correspond to the primary key of the other relation a "logical pointer"

If all foreign key constraints are enforced, *referential integrity* is achieved (i.e., no dangling references)

Foreign Keys in SQL

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

sid is a foreign key referring to Students

```
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	15-101	C
53666	18-203	B
53650	15-112	A
53666	15-105	B

Students

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

Enforcing Referential Integrity

Students and Enrolled; *sid* in Enrolled is a FK references Students

What to do if a tuple with a non-existent sid is inserted in Enrolled?

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 we can set sid to be equal to *null,* denoting *"unknown"* or *"inapplicable")*

Similar issues arise if primary key of Students tuple is updated

Integrity Constraints (ICs)

IC: 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 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!

Where do ICs Come From?

ICs are based upon the *real-world semantics*

we can check a database instance to see if an IC is violated, but we cannot infer that an IC hold

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

key and foreign key ICs are the most common

(more general ICs supported too)