The Relational Model

Intro & SQL overview

Keys & Integrity Constraints

ER to Relational

ISA to Relational
The Relational Model

Intro & SQL overview

Keys & Integrity Constraints

ER to Relational

ISA to Relational
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
<table>
<thead>
<tr>
<th>Relational</th>
<th>Key/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tables with rows and columns</td>
<td>collections of documents</td>
</tr>
<tr>
<td>well-defined schema</td>
<td>schema-less (each document can have different schema)</td>
</tr>
<tr>
<td>data model fits data rather than functionality</td>
<td>data stored in an application-friendly way</td>
</tr>
<tr>
<td>deduplication</td>
<td>possible duplication</td>
</tr>
</tbody>
</table>

based on a table from http://readwrite.com
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


*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

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

**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:

```sql
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):

```sql
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

ER to Relational

ISA to Relational
Keys

keys: associate tuples in different relations

keys are one form of integrity constraint (IC)

<table>
<thead>
<tr>
<th>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>cid</td>
</tr>
<tr>
<td>53666</td>
<td>15-101</td>
</tr>
<tr>
<td>53666</td>
<td>18-203</td>
</tr>
<tr>
<td>53650</td>
<td>15-112</td>
</tr>
<tr>
<td>53666</td>
<td>15-105</td>
</tr>
</tbody>
</table>

FOREIGN Key

PRIMARY Key
Primary Keys

A set of fields is a **superkey** if:
No two distinct tuples can have same values in all key fields

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

Is <sid> a superkey?
What about <sid,name>?
What about <sid,name,age>?
What about <age,name>?
Primary Keys

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

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>
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))

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

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>15-101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>18-203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>15-112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>15-105</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>
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 to us

key and foreign key ICs are the most common
(more general ICs supported too)
The Relational Model

Intro & SQL overview

Keys & Integrity Constraints

ER to Relational

ISA to Relational
Logical DB Design: ER to Relational

Entity sets to tables

CREATE TABLE Employees

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
</tr>
</tbody>
</table>
Relationship Sets to Tables

Our favorite example:

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ssn</th>
<th>did</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>51</td>
<td>1/1/91</td>
</tr>
<tr>
<td>123-22-3666</td>
<td>56</td>
<td>3/3/93</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>51</td>
<td>2/2/92</td>
</tr>
</tbody>
</table>
Relationship Sets to Tables

In translating a many-to-many relationship set to a relation, attributes of the relation must include:

- Keys for each participating entity set (as foreign keys). This set of attributes forms a superkey for the relation.
- All descriptive attributes.

```
CREATE TABLE Manages(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    PRIMARY KEY (ssn, did),
    FOREIGN KEY (ssn)
        REFERENCES Employees,
    FOREIGN KEY (did)
        REFERENCES Departments
)
```

<table>
<thead>
<tr>
<th>ssn</th>
<th>did</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>51</td>
<td>1/1/91</td>
</tr>
<tr>
<td>123-22-3666</td>
<td>56</td>
<td>3/3/93</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>51</td>
<td>2/2/92</td>
</tr>
</tbody>
</table>
Each dept has at most one manager, according to the *key constraint* on Manages.
Review: Key Constraints in ER

1-to-1

1-to Many

Many-to-1

Many-to-Many
Translating ER with Key Constraints

since each department has a unique manager, we could instead combine Manages and Departments

CREATE TABLE Manages(
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments)

Vs.

CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees)
What if the toy department has no manager (yet)?

Can be NULL!

CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn)
  REFERENCES Employees)
Review: Participation Constraints

does every employee work in a department? If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total (vs. partial)*.

Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)
Participation Constraints (PC) in SQL

PCs of one entity set in a binary relationship, yes! but little else (without resorting to CHECK constraints)

```sql
CREATE TABLE Dept_Mgr(
  did  INTEGER,
  dname  CHAR(20),
  budget  REAL,
  ssn  CHAR(11) NOT NULL,
  since  DATE,
  PRIMARY KEY  (did),
  FOREIGN KEY  (ssn) REFERENCES Employees,
  ON DELETE NO ACTION)
```
A **weak entity** can be identified uniquely by the primary key of another (**owner**) entity (+ some of its attributes)

- Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities)
- Weak entity set must have total participation in this **identifying** relationship set

---

**Employees**

- ssn
- name
- lot

**Policy**

- cost
- pname
- age

**Dependents**
Translating Weak Entity Sets

Weak entity set and identifying relationship set are translated into a single table.

When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE Dep_Policy (  
pname CHAR(20),  
age INTEGER,  
cost REAL,  
ssn CHAR(11) NOT NULL,  
PRIMARY KEY (pname, ssn),  
FOREIGN KEY (ssn) REFERENCES Employees,  
    ON DELETE CASCADE)
```
The Relational Model

Intro & SQL overview

Keys & Integrity Constraints

ER to Relational

ISA to Relational
Review: ISA Hierarchies

As in C++, or other PLs, attributes are inherited.

If we declare A **ISA** B, every A entity is also considered to be a B entity.

*Overlap constraints*: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? *(Allowed/disallowed)*

*Covering constraints*: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*
Translating ISA Hierarchies to Relations

CREATE TABLE Employees (  
  ssn CHAR(11) NOT NULL,  
  name CHAR(20),  
  lot INTEGER,  
  PRIMARY KEY (ssn))

CREATE TABLE Hourly_Emps (  
  ssn CHAR(11) NOT NULL,  
  hourly_wages REAL,  
  hours_worked REAL,  
  PRIMARY KEY (ssn),  
  FOREIGN KEY (ssn) REFERENCES Employees)

CREATE TABLE Contract_Emps (  
  ssn CHAR(11) NOT NULL,  
  contractid INTEGER,  
  PRIMARY KEY (ssn),  
  FOREIGN KEY (ssn) REFERENCES Employees)

what should happen if I delete an entry from Employees?  

can we use ON DELETE CASCADE?  

how to access name and hours worked?  

Join!
Alternative approach for ISA Hierarchies

CREATE TABLE Hourly_Emps (
  ssn  CHAR(11) NOT NULL,
  name CHAR(20),
  lot INTEGER,
  hourly_wages REAL,
  hours_worked REAL,
  PRIMARY KEY (ssn))

CREATE TABLE Contract_Emps (
  ssn CHAR(11) NOT NULL,
  name CHAR(20),
  lot INTEGER,
  contractid INTEGER,
  PRIMARY KEY (ssn))

how to ensure that every employee is only in one of the two?

what about Employees that are neither?

what about querying for all employees?

Query 2 tables!
Relational Model: Summary

Tabular representation of data
Simple & intuitive, currently the most widely used

Integrity Constraints can be specified based on app semantics & DBMS checks for violations
Two important ICs: primary and foreign keys
In addition, we *always* have domain constraints

ER to Relational is (fairly) straightforward