CSC 453
Database Technologies

Tanu Malik
DePaul University

A Data Model

- A notation for describing data or information.
- Consists of mostly 3 parts:
  - Structure of the data
    - Data structures and relationships
  - Operations on the data
    - programming operations
  - Constraints on the data
    - Describe limitations on data
  - Persistence of data
    - Data is stored permanently on disk

Data Models

- Models have evolved:
  - Files
  - Hierarchical
  - Network
  - Relational
  - Key-value
  - Graph
  - Document
  - Array/Matrix
  - Column

DBMS Evolution

- 1960s: Codasyl (network), IMS (hierarchical)
- 1970s: Relational Model (System R, Ingres)
- 1980: SQL as a language for RDBMS
- 1990: Object-Relational Model (disk-based databases)
- 2000's: NoSQL
- 2010: NewSQL (Main-memory databases)

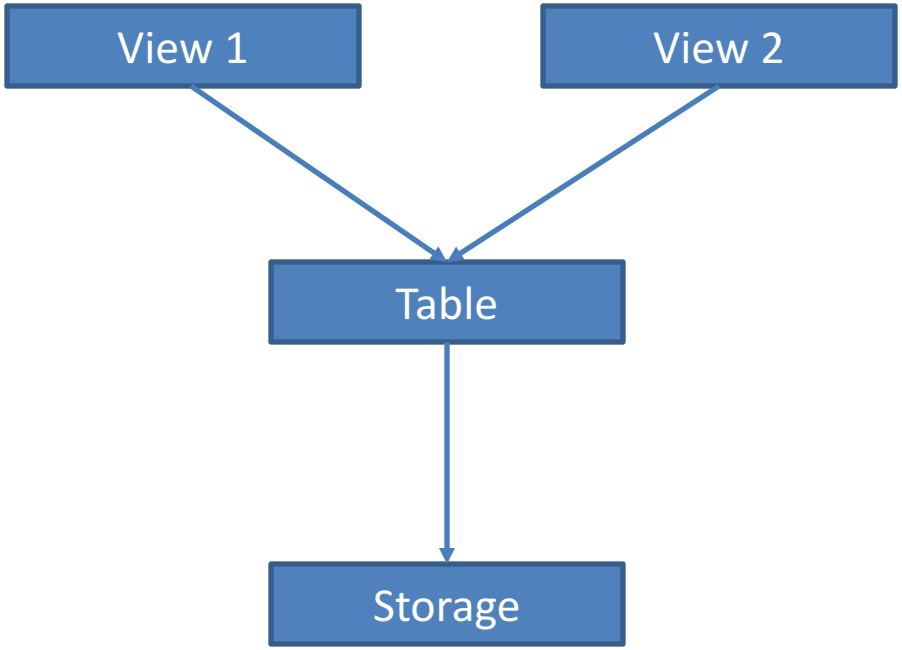
Relational Model

- Very popular
- Provides program-data independence
  - Create a 3 level architecture

Program-Data Independence

- View-Level
  - What data is exposed to users, what are they allowed to see
- Logical Level
  - Definition of tables, attributes, constraints, etc
- Physical Level
  - Data stored in files, location of files, indexed, storage layout

3-level architecture



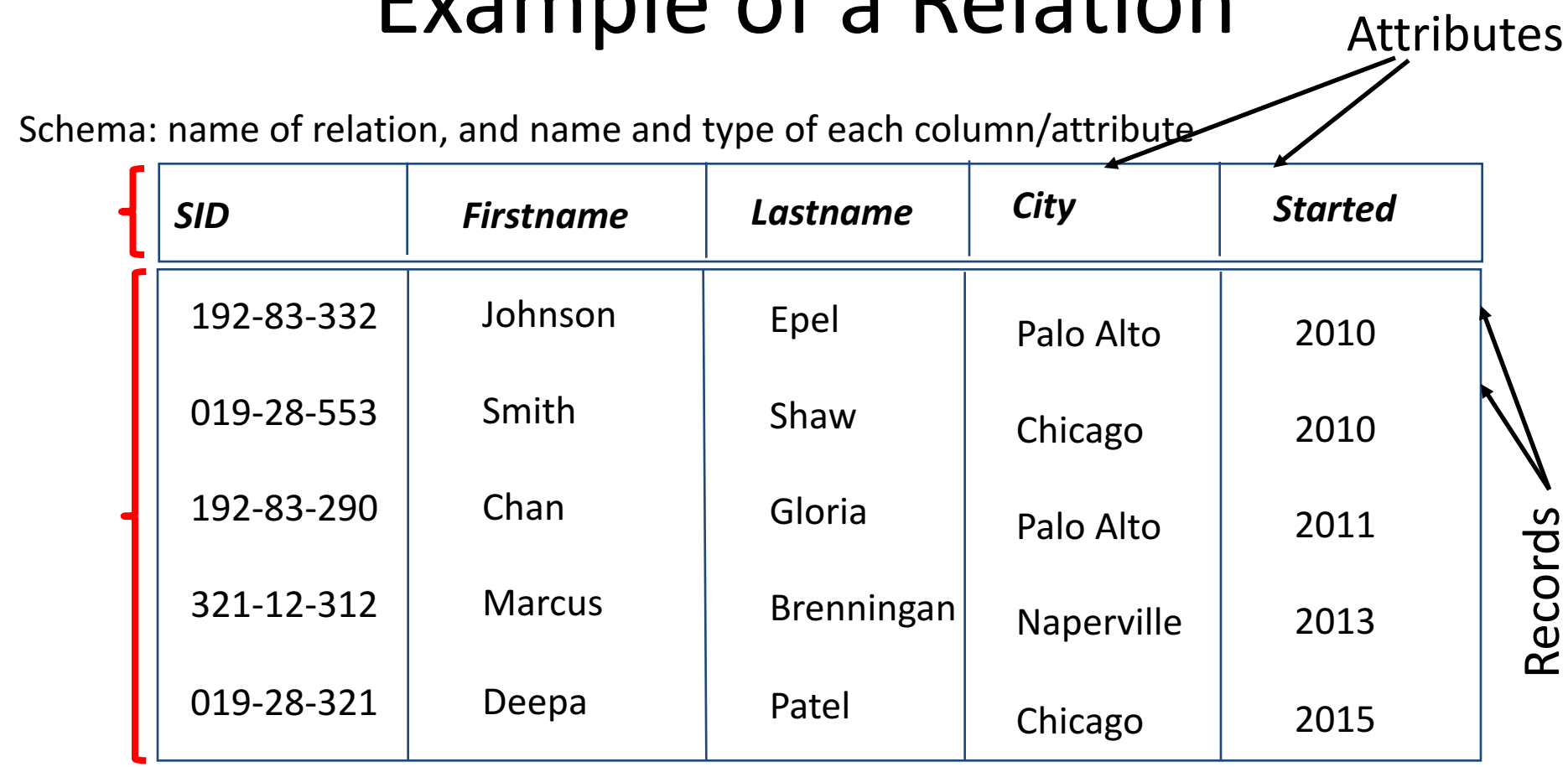
Program-Data Independence

- Logical Data Independence
  - Modify table definitions without changing view used in an application
    - Add/drop/rename attributes from a table
    - Rename a table
- Physical Data Independence
  - Modify the structure and organizaton of physical storage with changing logical schema
    - Change the type of index
    - Compress a table when it is stored on disk
    - Move a table to another disk/machine

Relational Model-Relation

- Data are divided into two-dimensional tables called *relations*, each representing some type of entity recorded in the database
- Each relation has a *name*, describing what entity it represents

Example of a Relation



Schema = **Student**(Sid, Firstname, Lastname, City, Started)

This whole table is an instance of the Student schema.

Properties of a Relation

- A relation is a set of records/tuples
  - All rows are distinct
  - Order of rows does not matter
  - What about columns (?)
- Values in a record
  - Each column has a *domain* from which its values are taken
  - A domain consists of *atomic* values – no multi-valued or divisible attributes are allowed
  - Missing/unknown values are indicated by NULL

Typical Domains

- Numerical values
  - integers, fixed-point values, floating-point values
- Character strings
  - fixed-length or variable-length
- Dates and times

Relation-Mathematical Equivalence

- A binary relation on a set A is a collection of ordered pairs of elements of A.
  - It is subset of cartesian product $A^2 = A \times A$

- A database relation is a n-ary relation on a set A with attributes $A_1, A_2, \ldots, A_n$
  - Data in a database relation is a subset of $A_1 \times A_2 \times \ldots \times A_n$

Relational Model-Implementation

SQL- A language for relational DBs

- Data Definition Language (DDL)
  - Create, modify, delete relations
  - Specify constraints
  - Administer users, security, etc

Creating a Table

- CREATE TABLE *TABLE_NAME*
  (*Attribute1 DOMAIN_1 ,*
  Attribute2 DOMAIN_2 ,
  …
  AttributeN DOMAIN_N);
- You can list as many attributes and domains as you want, separated by commas

SQL Domains

- Numerical domains:
  - Postgres: Integer
  - Oracle:
    - NUMBER: A general floating-point number (also REAL, FLOAT)
    - NUMBER(*, 0): A general integer (also INTEGER, INT)
    - NUMBER(*n*): An *n*-digit integer
    - NUMBER(*x*, *y*): A fixed-precision number with *x* total digits, and *y* digits to the right of the decimal point (also DECIMAL, NUMERIC)

SQL Domains

- String domains:
  - CHAR(*n*): A fixed-length string of *n* characters
  - VARCHAR(*m*) or VARCHAR2(*m*): A variable-length string of up to *m* characters
- Dates:
  - DATE: A date in ‘dd-mon-yy’ format (dd = day, mon = month, yy = year)
  - (There are often variations in the date format in different SQL implementations…)

Populating a Table

- To insert a record into a table:

  INSERT INTO *TABLE_NAME*
  VALUES (*value1, value2, value3, ...*);

- Values of attributes must be given in the same order as in the schema

Example

```
create table student (
  LastName      varchar(40),
  FirstName     varchar(40),
  SID           number(5),
  SSN           number(9),
  Career        varchar(4),
  Program       varchar(10),
  City          varchar(40),
  Started       number(4)
  );
```

Example

```
insert into student
    values ( 'Brennigan', 'Marcus', 90421, 987654321,
'UGRD', 'COMP-GAM', 'Evanston', 2010 );
insert into student
    values ( 'Patel', 'Deepa', 14662, null, 'GRD', 'COMP-
SCI', 'Evanston', 2013 );
insert into student
    values ( 'Snowdon', 'Jonathan', 08871, 123123123,
'GRD', 'INFO-SYS', 'Springfield', 2009 );

insert into course
    values ( 1020, 'Theory of Computation', 'CSC', 489);
insert into course
    values ( 1092, 'Cryptography', 'CSC', 440);
insert into course
    values ( 3201, 'Data Analysis', 'IT', 223);
```

Record Course Data

"CID"	"COURSENAME"	"DEPARTMENT"	"COURSENR"
"1020"	"Theory of Computation"	"CSC"	"489"
"1092"	"Cryptography"	"CSC"	"440"
"3201"	"Data Analysis"	"IT"	"223"
"9219"	"Databases Systems"	"CSC"	"355"
"3111"	"Theory of Computation"	"CSC"	"389"
"8772"	"History of Games"	"GAM"	"206"
"2987"	"Topics in Digital Cinema"	"DC"	"270"

Example

```
create table course (
  CID   number(4),
  CourseName    varchar(40),
  Department    varchar(4),
  CourseNr       char(3)
);
```

Exercise

CREATE TABLE movie(
mid integer,
title varchar(50),
year integer
director name varchar(50) not null,
language varchar(30),
primary key (mid),
foreign key director name references director,
foreign key (leading actor) references actor

Deleting Tables

- Deleting entire table:
  - `drop table student;`
- Deleting specific rows:
  - `delete from student where name = 'Smith';`

Record Enrollments

"STUDENTID"	"COURSEID"	"QUARTER"	"YEAR"
"11035"	"3201"	"Fall"	"2015"
"11035"	"1020"	"Fall"	"2012"
"11035"	"1092"	"Fall"	"2012"
"75234"	"3201"	"Winter"	"2012"
"8871"	"1092"	"Fall"	"2013"
"39077"	"1092"	"Fall"	"2013"
"57923"	"9219"	"Winter"	"2013"
"19992"	"3201"	"Winter"	"2013"

Example

```
create table enrolled(
  StudentID  number(7),
  CourseID   number(4),
  Quarter    varchar(40),
  Year    varchar(4)
);
```

Constraints

Constraints

- Constraints are maintained by the DBMS:
  - Domain constraints
    - All values of each attribute must come from the specified domain
  - Custom constraints
    - E.g. account balance must be positive
  - Integrity constraints
    - Uniqueness, association

Defaults and Constraints

- After attribute and domain, before comma:
  - Add default value for the attribute with DEFAULT *value*
  - Disallow NULL values with NOT NULL
  - Force values to be unique with UNIQUE
  - Impose other constraints with CHECK (*condition*)
    - e.g., CHECK (*low <= Attribute AND Attribute <= high*)
- Applied whenever a tuple is added/modified

Example

```
create table student (
  LastName      varchar(40) not null,
  FirstName     varchar(40),
  SID           number(5),
  SSN           number(9) unique,
  Career        varchar(4),
  Program       varchar(10),
  City          varchar(40),
  Started       number(4)
  );
```

Primary Keys

- A set of attributes in the relation may be defined to be the *primary key* – it must be a minimal set of attributes that uniquely identifies each tuple in the relation
- NULL values are not allowed in the primary key of a relation (“entity integrity”)
- Primary Key = = Unique, Not Null
- The primary key is underlined in the schema

Example

```
create table course (
  CID   number(4),
  CourseName    varchar(40),
  Department    varchar(4),
  CourseNr       char(3),

  primary key (CID)
);
```

Example

```
create table student (
  LastName      varchar(40) not null,
  FirstName     varchar(40),
  SID           number(5),
  SSN           number(9),
  Career        varchar(4),
  Program       varchar(10),
  City          varchar(40),
  Started       number(4),

  primary key (SID),
  unique(SSN)
);
```

Exercise

```
create table student (
  LastName      varchar(40),
  FirstName     varchar(40),
  SID           number(5),
  SSN           number(9),
  Career        varchar(4),
  Program       varchar(10),
  City          varchar(40),
  Started       number(4),

  primary key (SID),
  unique(SSN)
);
```

```
create table student (
  LastName      varchar(40),
  FirstName     varchar(40),
  SID           number(5),
  SSN           number(9),
  Career        varchar(4),
  Program       varchar(10),
  City          varchar(40),
  Started       number(4),

  primary key (SID, SSN),
);
```

Exercise

```
create table student (
  LastName      varchar(40),
  FirstName     varchar(40),
  SID           number(5),
  SSN           number(9)
not null,
  Career        varchar(4),
  Program       varchar(10),
  City          varchar(40),
  Started       number(4),

  primary key (SID),
  unique(SSN)
);
```

```
create table student (
  LastName      varchar(40),
  FirstName     varchar(40),
  SID           number(5),
  SSN           number(9),
  Career        varchar(4),
  Program       varchar(10),
  City          varchar(40),
  Started       number(4),

  primary key (SID, SSN),
);
```

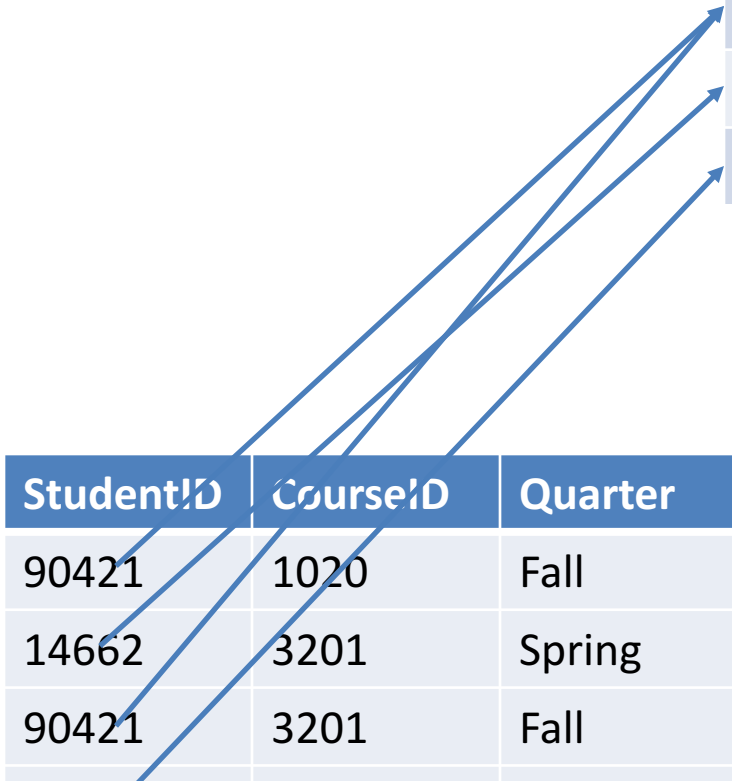
Foreign Keys

- Set of fields `refering' to a tuple in another relation
  - Must correspond to the primary key of other relation
  - Like a `logical pointer'

Foreign Keys

SID	Lastname	Firstname	SSN
90421	Brennigan	Marcus	987654321
14662	Patel	Deepa	NULL
08871	Snowdon	Jon	123123123

StudentID	CourseID	Quarter	Year
90421	1020	Fall	2016
14662	3201	Spring	2016
90421	3201	Fall	2016
08871	2987	Fall	2016



Example

```
create table enrolled (
  StudentID     number(5),
  CourseID      number(4),
  Quarter       varchar(6),
  Year          number(4),

  primary key (StudentID, CourseID),
  foreign key (StudentID) references student(SID),
  foreign key (CourseID) references course(CID)
);
```

Exercise

- Create table for Studentgroups
  - Groupname, President, Founded
  - President of a SG is a student
  - Studentgroups are unique
- Create table Members
  - Students are members of groups
  - Students join groups in a given year

Enforcing Integrity Constraints

Enforcing Referential Integrity

- Can only insert or update a tuple if the value of every foreign key in the tuple appears among the values of the primary key that it references

SID	Lastname	Firstname	SSN
90421	Brennigan	Marcus	987654321
14662	Patel	Deepa	NULL
08871	Snowdon	Jon	123123123

StudentID	CourseID	Quarter	Year
90421	1020	Fall	2016
14662	3201	Spring	2016
90421	3201	Fall	2016
08871	2987	Fall	2016

Insert (40563, 1020, 'Fall', `2016'):
Reject such an insertion

- Can only delete or update a tuple if the value of its primary key does not appear among the values of any of the foreign keys that reference it

SID	Lastname	Firstname	SSN
90421	Brennigan	Marcus	987654321
14662	Patel	Deepa	NULL
08871	Snowdon	Jon	123123123

StudentID	CourseID	Quarter	Year
90421	1020	Fall	2016
14662	3201	Spring	2016
90421	3201	Fall	2016
08871	2987	Fall	2016

Delete from Student where SID = 90421
Reject such a deletion

Handling Violations

- Operations that violate constraints are by default rejected, but there are other options:
  - Prompt user to change offending value(s)
  - Use default value in place of offending value(s)
  - Use NULL in place of offending value(s)
  - Allow operation, but cascade changes from primary keys to foreign keys as needed

- We can specify actions if referential integrity of a foreign key is violated

```
SET NULL
SET DEFAULT
CASCADE
```

- Specified as

```
ON DELETE/UPDATE
    SET NULL/CASCADE/DEFAULT
```

Forcing Deletions

- CASCADE Constraints
  - Remove referencing tuples before referenced tuples

```
create table enrolled (
  StudentID     number(5),
  CourseID      number(4),
  Quarter       varchar(6),
  Year          number(4),

  primary key (StudentID, CourseID),
  foreign key (StudentID) references student(SID),
  foreign key (CourseID) references course(CID)
  on delete cascade
);
```

Referential triggered action

```
Example (CASCADE)
CREATE TABLE dependent
( ...
FOREIGN KEY (essn) REFERENCES employee(ssn)
ON DELETE CASCADE,
...)

Example (SET NULL)
CREATE TABLE studentgroup (
FOREIGN KEY (PresidentID) REFERENCES student(SID)
ON DELETE SET NULL

Example (SET DEFAULT)
CREATE TABLE employee (
...
dno INT NOT NULL DEFAULT 1,
...
FOREIGN KEY (dno) REFERENCES department(dnumber)
ON DELETE SET DEFAULT ...)
```

Enforcing Integrity Constraints-Summary

- Integrity constraints are specified when schema is defined
- They must be true at all times
  - Must be checked when relations are modified
  - A DBMS must be responsible for checking them (as opposed to?)
- Integrity constraints come from applications; a DB instance is never a good evidence to infer ICs

Relation Modifications

UPDATE

UPDATE *TABLE_NAME*
SET *Attribute = value*
WHERE *condition;*

- Sets *Attribute* to *value* in exactly those tuples that satisfy *condition*

DELETE

DELETE FROM *TABLE_NAME*
WHERE *condition*;

- Removes from the table exactly those tuples that satisfy *condition*

ALTER TABLE

ALTER TABLE *TABLE_NAME …*
ADD *Attribute DOMAIN*;

or

DROP COLUMN *Attribute*
CASCADE CONSTRAINTS;

- Modifies an existing table schema

Examples

```
ALTER TABLE student
ADD age integer;
```

Exercise: Add a (named) constraint that 0 <= age <= 120

```
ALTER TABLE studentgroup
ADD FOREIGN KEY(PresidentID) REFERENCES Student(SID);

ALTER TABLE studentgroup
ADD CONSTRAINT fk_sg
FOREIGN KEY(PresidentID) REFERENCES Student(SID);

ALTER TABLE studentgroup DROP fk_sg;
```

Cyclic Dependencies

- Most systems do not allow references to tables that do not exist yet.
- Two solutions:
  - if no cyclical dependencies: create tables in right order (Example: university.sql)
- in case of cyclical dependencies: create tables without f.k. constraints, and use ALTER TABLE to add these later

Exercise

- Design a relation schema on Social Networks with the following requirements:
  - Every user of a social network is identified by user_id, username, email and password.
  - Users are in relationships with other users with a current status.