CPSC 304
Introduction to Database Systems

The Relational Model (Part 1 of 2)

Fall 2017

References:

Database Management Systems, by Ramakrishnan and Gehrke (Chapter 3, Sections 3.1-3.5 of the Textbook)

Acknowledging previous work by CPSC 304 instructors over the years (alphabetically: Ed, George, Hassan, Hazra, Laks, Rachel, Raymond)

Learning Goals

- Compare and contrast *logical* and *physical data independence*.
- Define the components (and synonyms) of the relational model: tables, rows, columns, keys, associations, etc.
- Create tables, including the attributes, keys, and field lengths, using SQL's Data Definition Language (DDL).
- Explain and differentiate the kinds of integrity constraints in a database.
- Explain the purpose of referential integrity.
- Enforce referential integrity in a database using SQL's Data Manipulation Language (DML). Determine which delete, insert, or update policy to use when coding rules/defaults for referential integrity. Analyze the impact that a poor choice has.
- Map ER diagrams to the relational model (i.e., DDL), including constraints, weak entity sets, etc.

Databases – The Continuing Saga

- So far we've learned that databases are handy for many reasons.
- Before we can use them, we must design them.
- Previously, we saw how to use ER diagrams to design the *conceptual schema,* but we need to store the data.
- In this unit, we'll learn to use a *logical schema* to actually store the data. We'll be using the *relational model*.

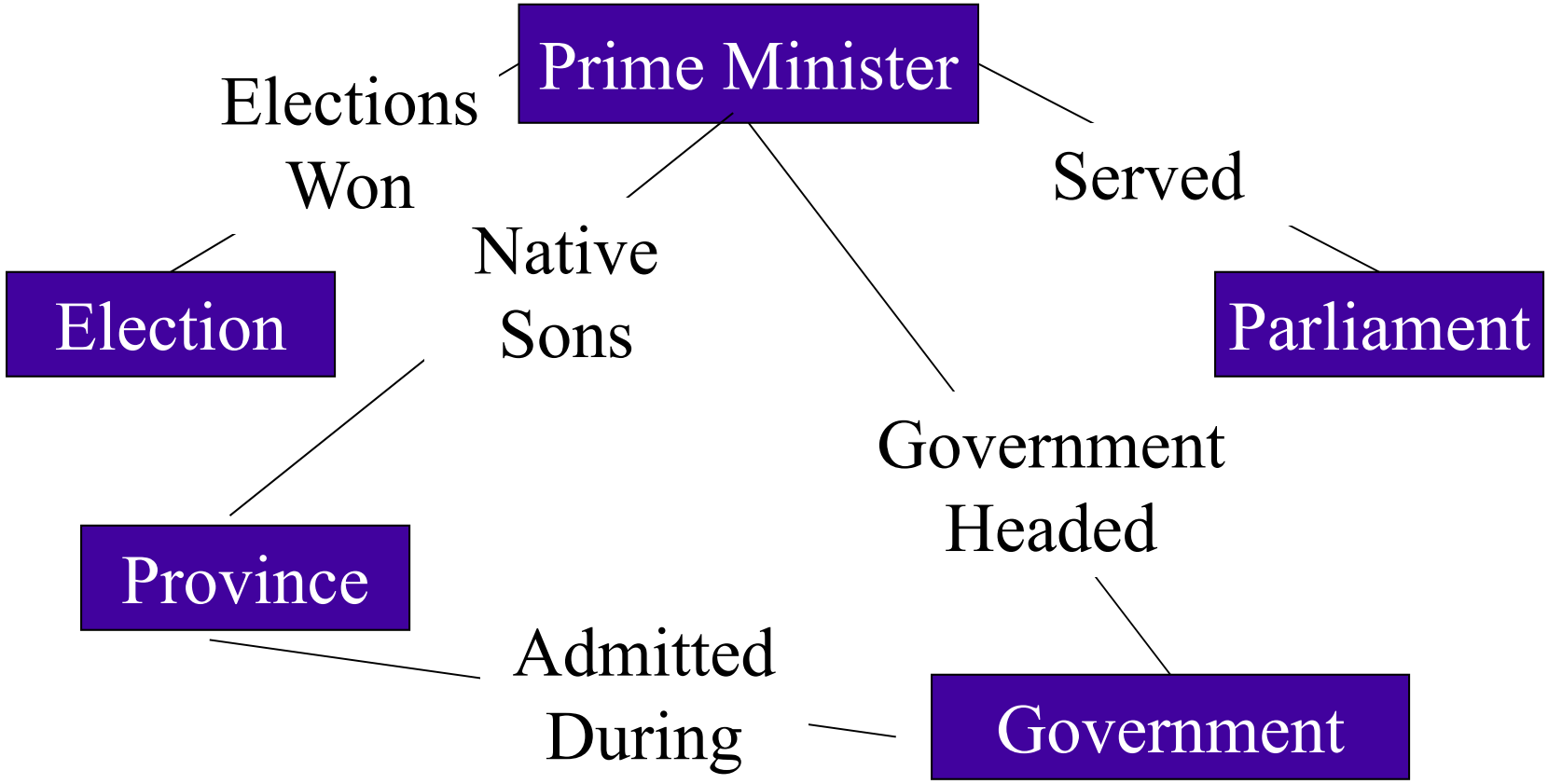
What Do We Want Out of Our Logical Schema Representation?

- The ability to store data without worrying about the blocks on disk—save that for CPSC 404
- The ability to query data easily
- A representation that is easy to understand
- A representation that we can easily adapt from the conceptual schema
- Separation from application programming language(s)

How Did We Get the Relational Model?

- Prior to the relational model, there were two main contenders for DBMSs:
  - Network database systems
  - Hierarchical database systems
- Network databases had a complex data model
  - Lots of pointers to follow around
  - Not very flexible
- Hierarchical databases integrated the application in the data model
  - Easier to understand than network databases (intuitive for many applications)

Example of the Hierarchical Model



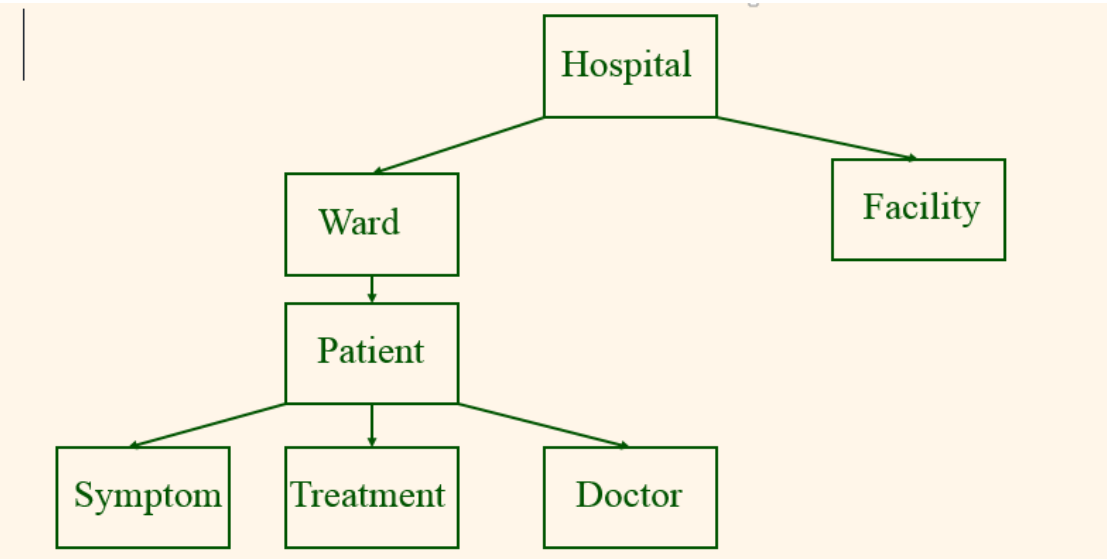
Looks similar to ER diagrams,
but has fewer concepts ... but let's see how you query it …

Example IMS (Hierarchical) Query: Print the names of all the provinces admitted during a Liberal Government

```
DLITPLI:PROCEDURE (QUERY_PCB) OPTIONS (MAIN);

 DECLARE QUERY_PCB POINTER;
 /*Communication Buffer*/
 DECLARE 1 PCB BASED(QUERY_PCB),
   2 DATA_BASE_NAME CHAR(8),
   2 SEGMENT_LEVEL CHAR(2),
   2 STATUS_CODE CHAR(2),
   2 PROCESSING_OPTIONS CHAR(4),
   2 RESERVED_FOR_DLI FIXED BINARY(31,0),
   2 SEGMENT_NAME_FEEDBACK CHAR(8)
   2 LENGTH_OF_KEY_FEEDBACK_AREA FIXED BINARY(31,0),
   2 NUMBER_OF_SENSITIVE_SEGMENTS FIXED BINARY(31,0),
   2 KEY_FEEDBACK_AREA CHAR(28);
 /* I/O Buffers*/
 DECLARE PRES_IO_AREA CHAR(65),
   1 PRESIDENT DEFINED PRES_IO_AREA,
   2 PRES_NUMBER CHAR(4),
   2 PRES_NAME CHAR(20),
   2 BIRTHDATE CHAR(8)
   2 DEATH_DATE CHAR(8),
   2 PARTY CHAR(10),
   2 SPOUSE CHAR(15);
 DECLARE SADMIT_IO_AREA CHAR(20),
   1 province_ADMITTED DEFINED SADMIT_IO_AREA,
   2 province_NAME CHAR(20);
 /* Segment Search Arguments */
 DECLARE 1 PRESIDENT_SSA STATIC UNALIGNED,
   2 SEGMENT_NAME CHAR(8) INIT('PRES '),
   2 LEFT_PARENTHESIS CHAR (1) INIT('('),
   2 FIELD_NAME CHAR(8) INIT ('PARTY '),
   2 CONDITIONAL_OPERATOR CHAR (2) INIT('='),
   2 SEARCH_VALUE CHAR(10) INIT ('Liberal '),
 2 RIGHT_PARENTHESIS CHAR(1) INIT(')');
   DECLARE 1 province_ADMITTED_SSA STATIC UNALIGNED,
     2 SEGMENT_NAME CHAR(8) INIT('SADMIT ');
   /* Some necessary variables */
   DECLARE GU CHAR(4) INIT('GU '),
     GN CHAR(4) INIT('GN '),
     GNP CHAR(4) INIT('GNP '),
     FOUR FIXED BINARY (31) INIT (4),
     SUCCESSFUL CHAR(2) INIT(' '),
     RECORD_NOT_FOUND CHAR(2) INIT('GE');
   /*This procedure handles IMS error conditions */
   ERROR;PROCEDURE(ERROR_CODE);
    *
    *
    *
   END ERROR;
   /* Main Procedure */
   CALL PLITDLI(FOUR,GU,QUERY_PCB,PRES_IO_AREA,PRESIDENT_SSA);
   DO WHILE(PCB.STATUS_CODE=SUCCESSFUL);
     CALL PLITDLI(FOUR,GNP,QUERY_PCB,SADMIT_IO_AREA,province_ADMITTED_SSA);
     DO WHILE(PCB.STATUS_CODE=SUCCESSFUL);
       PUT EDIT(province_NAME)(A);
     CALL PLITDLI(FOUR,GNP,QUERY_PCB,SADMIT_IO_AREA,province_ADMITTED_SSA);
       END;
     IF PCB.STATUS_CODE NOT = RECORD_NOT_FOUND
       THEN DO;
         CALL ERROR(PCB.STATUS_CODE);
         RETURN;
         END;
       CALL PLITDLI(FOUR,GN,QUERY_PCB,PRES_IO_AREA,PRESDIENT_SSA);
     END;
     IF PCB.STATUS_CODE NOT = RECORD_NOT_FOUND
       THEN DO;
         CALL ERROR(PCB.STATUS_CODE);
         RETURN;
         END;
 END DLITPLI;
```

Simpler Example: Structure of an IMS Query



- *Get Unique:* Ward (HNAME = "LIONS GATE", WNAME = "ICU")
- Until failure do:
  - *Get Next*: Patient (HNAME = "LIONS GATE", WNAME = "ICU", AGE < 40)
    - Until failure do:
      - *Get Next within Parent*: Symptom

Reference: *IMS Programming Techniques: A Guide to Using DL/I*, by Kapp & Leben, 1978

Example, IMS: Types of DL/I Calls

FYI only (don't memorize this IMS stuff) …

- DL/I = Data Language/1
  - IMS DB calls that are embedded in application programs
  - Similar to embedded SQL, though more complicated
- Segment = well-defined groupings of data elements (fields)
  - Compare to a *relation* in an RDBMS
- GU starts the search at the root
  - Retrieves the first segment that meets the *segment search arguments* (SSAs), if any SSAs are present
  - Sets *parentage* and *position*
- GN/GNP starts the search from your current position in the database
  - Retrieves the next segment that meets SSAs (if any)
  - GN sets parentage and position; but GNP only sets position

IMS: Types of DL/I Calls (cont.)

- Other calls: GHU, GHN, GHNP, ISRT, DLET, REPL, CHKP…
  - GHU means “Get Hold Unique”
    - To delete the current segment, DLET must be your very next call; else you lose the “hold”
  - DLET also deletes all descendants of a node
    - This could be problematic.
    - Later, compare to referential integrity rules in an RDBMS (e.g., CASCADE DELETE).
  - Later, compare DL/I calls to (simpler!) SQL calls.

IMS Example: Hospital Database DBD

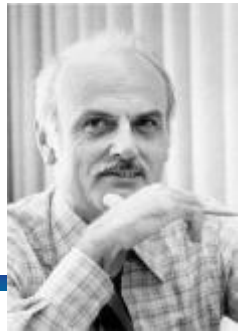
(based on Kapp/Leben)

```
PRINT       NOGEN
DBD         NAME=HOSPDBD, ACCESS=HISAM
DATASET     DD1=PRIME,OVFLW=OVERFLW,DEVICE=3330
SEGM        NAME=HOSPITAL,PARENT=0,BYTES=60
FIELD       NAME=(HNAME,SEQ,U), BYTES=40,START=1,TYPE=C
SEGM        NAME=WARD,PARENT=HOSPITAL,BYTES=31
FIELD       NAME=(WNAME,SEQ,U),BYTES=30,START=1,TYPE=C
FIELD       NAME=BEDAVAIL,BYTES=3,START=31,TYPE=C
…           …
SEGM        NAME=PATIENT,PARENT=WARD,BYTES=125
FIELD       NAME=(BEDIDENT,SEQ,U),BYTES=4,START=71,TYPE=C
FIELD       NAME=PATNAME,BYTES=30,START=1,TYPE=C
…           …
SEGM        NAME=SYMPTOM,PARENT=PATIENT,BYTES=77
FIELD       NAME=(SYMPDATE,SEQ),BYTES=16,START=21,TYPE=C
…           …
DBDGEN
FINISH
END
```

Program Communication Blocks (PCBs)

- A Program Specification Block (PSB) contains one or more Program Communication Blocks (PCBs)
- A PCB is the interface between a program and IMS, and keeps track of:
  - *Position* in the DB
  - *Status Code* returned (very important, check after each call)
    - e.g., “ ” (success), GE (empty), GB (reached end of DB), GA (for Get Next calls: segment found, but is ancestor of previous segment), GK (… found, but sibling is a different segment type), II (for ISRT calls: duplicate segment), etc.
  - *Key Feedback Area* (i.e., *concatenated key* of each segment in the hierarchical path up to and including the current segment)
    - e.g., “Lions Gate ICU 103759 20070829”
- Each DL/I call must specify which PCB it’s using in the PSB.

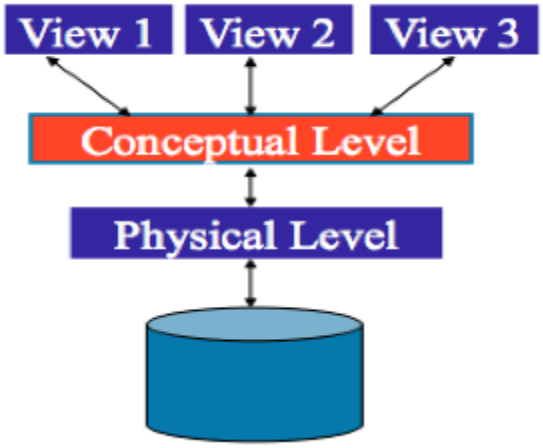
Relational Model to the Rescue!



- Introduced by Edgar Codd (IBM) in 1970
- Most widely used model today
  - Vendors: IBM, Microsoft, Oracle, Sybase, MySQL (open source), etc.
- Competitor: Object-oriented model
  - ObjectStore, Versant, Ontos
  - A synthesis emerging: *Object-relational model*
    - Informix Universal Server (Informix was purchased by IBM), UniSQL, O2, Oracle, DB2
- Recent competitors (triggered by the needs of Web):
  - XML data model
  - NoSQL

Key Points of the Relational Model

- Very simple to understand
  - Main abstraction is represented as a table
- Underlying mathematical foundations—sets, rel. algebra
- **Physical Data Independence**
  - Ability to modify the physical schema without changing the logical schema
- **Logical Data Independence** – done with views
  - Ability to change the conceptual schema without changing the application



Structure of Relational Databases

- **Relational database**: a set of **relations**
- **Relation:** made up of 2 parts:
  - **Schema**: specifies name of relation, plus name and **domain** (type) of each **attribute**.
    - e.g., Student (sid: string, name: string, address: string, phone: string, major: string)
  - **Instance**: a **table** with **rows** and **columns**
    - **# of rows = cardinality**
    - **# of columns = arity or degree**
- **Relational Database Schema:** a collection of schemas in the database
- **Database Instance:** a collection of instances of its relations

Example of a Relation Instance

attribute, column name

relation name

Student

sid	name	address	phone	major
99111120	G. Jones	1234 W. 12th Ave., Van.	889-4444	CPSC
92001200	G. Smith	2020 E. 18th St., Van	409-2222	MATH
94001020	A. Smith	2020 E. 18th St., Van	222-2222	CPSC
94001150	S. Wang	null	null	null

tuple, row, record

domain value

- degree/arity = 5; cardinality = 4,
- Order of rows isn't important
- Order of attributes isn't important (except in some query languages)

Formal Structure

- Formally, a relation *r* is a set (a_1, a_2,…,a_n) where a_i is in D_i, the domain (set of allowed values) of the *i*-th attribute.
- Attribute values are atomic, i.e., integers, floats, strings
- A domain contains a special value, **null**, indicating that the value is not known (or not applicable such as in the case of a missing cell phone number).
- If A_1,…, A_n are attributes with domains D_1,..., D_n, then
  (A_1:D_1, …, A_n:D_n)
  is a **relation schema** that defines a relation type.
  Sometimes we leave off the domains.
- Student (*sid*: string, *name*: string, *address*: string, *phone*: string, *major*: string)

Example of a Formal Definition

Student

sid	name	address	phone	major
99111120	G. Jones	1234 W. 12th Ave., Van.	889-4444	CPSC
92001200	G. Smith	2020 E. 18th St., Van	409-2222	MATH
94001020	A. Smith	2020 E. 18th St., Van	222-2222	CPSC
94001150	S. Wang	null	null	null

Student (sid: integer, name: string, address: string,
phone: string, major: string)

Or, without the domains, simply:

Student (sid, name, address, phone, major)

Clicker Question

- Here is a table representing a relation named R. Identify the attributes, schema, and tuples of R. Which of the following is not a true statement about R?

A	B	C
0	1	2
3	4	5
6	7	8
9	10	11

A. R has four tuples.
B. B is an attribute of R.
C. (6,7,8) is a tuple of R.
D. The schema of R is R(A,B,C).
E. None of the above

Relational Query Languages

- A major strength of the relational model is that results in simple but powerful *querying* of data.
- Queries can be written intuitively.
- The DBMS's optimizer is responsible for efficient evaluation (see CPSC 404). A user, programmer, or DBA usually doesn't do this.
  - There are precise semantics for relational queries.
  - The optimizer can extensively re-order operations, while ensuring that the answer does not change.

Raymond Boyce

Don Chamberlain

Structured Query Language (SQL)

- Developed by IBM (System R) in the 1970s
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, current standard)
  - SQL-99 (major extensions)

A Peek at the SQL Query Language

Student

sid	name	address	phone	major
99111120	G. Jones	1234 W. 12th Ave., Van.	889-4444	CPSC
92001200	G. Smith	2020 E. 18th St., Van	409-2222	MATH
94001020	A. Smith	2020 E. 18th St., Van	222-2222	CPSC

- Find the id's, names, and phone #'s of all CPSC students:

```
SELECT  sid, name, phone
FROM    Student
WHERE   major = "CPSC"
```

sid	name	phone
99111120	G. Jones	889-4444
94001020	A. Smith	222-2222

■ To select the whole row of the relation, simply replace "SELECT sid, name, phone" with "SELECT * ".

A Peek at the SQL Query Language (cont.): Querying Multiple Tables

Student

sid	name	address	phone	major
99111120	G. Jones	…	…	CPSC
…	…	….	…	…

Grade

sid	dept	course#	mark
99111120	CPSC	122	80
…	…	….	…

- To select id and names of the students who have taken some CPSC course, we write:

```
SELECT    sid, name
FROM      Student, Grade
WHERE     Student.sid = Grade.sid AND dept = 'CPSC'
```

Compare with the Hierarchical Example

Simple, eh?

- We'll see more about how to query tables in Chapter 5 when we study SQL's Data Manipulation Language (DML).
- Note that you can't query anything without having a place to store your data; so, let's go back and see how we create relations. For this, we'll use SQL's Data Definition Language (DDL).
- Later: There's also SQL's Data Control Language (DCL) to manage security.

Creating Relations in SQL/DDL

- The statement on the right creates the Student relation.
  - The type (domain) of each attribute is specified, and is enforced whenever tuples are added or modified.

```
CREATE TABLE Student
        (sid       INTEGER,
         name      CHAR(20),
         address   CHAR(30),
         phone     CHAR(13),
         major     CHAR(4) )
```

- The statement on the right creates the Grade relation about courses that a student takes.

```
CREATE TABLE Grade
        (sid       INTEGER,
         dept      CHAR(4),
         course#   CHAR(3),
         mark      INTEGER)
```

Dropping and Altering Relations

```
DROP TABLE Student
```

- Deletes (drops) the Student relation
  - The schema information *and* its tuples (data) are deleted.

real number

```
ALTER TABLE Student
        ADD COLUMN gpa REAL;
```

- The schema of Students is altered by adding a new attribute.
  - Every tuple in the current instance is extended with a **null** value in the new attribute. Thus, existing data is preserved.

Adding and Deleting Tuples

- We can insert a new tuple:

```
INSERT
INTO     Student (sid, name, address, phone, major)
VALUES   ('52033688', 'G. Chan', '1235 W. 33, Van',
          '882-4444', 'PHYS')
```

- We can delete all tuples satisfying some condition (e.g., name = 'Smith'):

```
DELETE
FROM     Student
WHERE    name = 'Smith'
```

Powerful variants of these commands exist; more later.

Integrity Constraints (ICs)

- **IC:** a 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.
  - The DBMS should not allow illegal instances.
  - This avoids data entry errors, too.
- The types of ICs depend on the data model.
  - What did we have for ER diagrams?
  - Next up: Constraints for relational databases

Keys Constraints (for Relations)

- Similar to those for entity sets in the ER model
- A set S = {S_1, S_2, …, S_m} of attributes in an *n*-ary relation ($1 \leq m \leq n$) is a **key** (or ***candidate key***) for a relation if:
  1. No distinct tuples can have the same values in all key attributes, and
  2. No subset of S is itself a key (according to (1)).
     - If such a subset exists, then S is a **superkey** and not a key.

sueprkey is not necessary a key

- One of the possible keys is chosen to be the **primary key** (PK).
- e.g.
  - {*sid, name*} is a superkey
  - ***sid*** is the PK for Student

```
CREATE TABLE Student
(sid      INTEGER  PRIMARY KEY,
 name     CHAR(20),
 address  CHAR(30),
 phone    CHAR(13),
 major    CHAR(4))
```

Keys Constraints in SQL

- A **primary key constraint** specifies a table's primary key.
  - A primary key attribute cannot be *null.*
- Other keys are specified using a **unique key constraint.**
  - This means the values for a group of attributes must be unique (if they are not null).
  - However, these attributes can be *null.*
- Key constraints are checked when:
  - New values are inserted
  - Their values are modified.

Keys Constraints in SQL (cont.)

- (Example 1) "For a given student and course, there is a single grade." No student can take the same course twice.

```
CREATE TABLE Grade
  (sid       INTEGER,
   dept      CHAR(4),
   course#   CHAR(3),
   mark      INTEGER,
   PRIMARY KEY (sid, dept, course#))
```

- (Ex. 2 - Silly) "Students can take a course once, and receive a single grade for that course; but, no two students in a course can receive the same grade."

```
CREATE TABLE Grade2
  (sid       INTEGER,
   dept      CHAR(4),
   course#   CHAR(3),
   mark      INTEGER,
   PRIMARY KEY (sid, dept, course#),
   UNIQUE (dept, course#, mark) )
```

Single attribute keys can also be declared as keys on the same line as the attribute.

Clicker Question

- Consider the table definition: CREATE TABLE Employee (

Numbers denote lines only →

```
(1) id          INTEGER,
(2) sin         INTEGER,
(3) name        CHAR(20),
(4) managerID   INTEGER );
```

- Which of the following is **not** a legal addition?

A. Add UNIQUE just before the commas on lines (2) and (3) and add PRIMARY KEY just before the comma on line (1).

B. Add PRIMARY KEY just before the commas on lines (1) and (2).

C. Add UNIQUE just before the comma on line (1), and add PRIMARY KEY just before the comma on line (2).

D. All are legal

E. None are legal

Foreign Key Constraints

- **Foreign Key (FK)**: A set of attributes in one relation used to 'reference' a unique tuple in another relation.
  - A foreign key must correspond to a candidate key in the other relation, but usually it's the primary key of that other relation.
  - Like a 'logical pointer'.
- For example, there are two FKs for:
  Grade (sid, dept, course#, grade)
  - sid is an FK referring to a key value in the Student relation
  - (dept, course#) is a composite FK, referring to the Course relation
- **Referential Integrity:** All foreign keys must reference existing entities.
  - There are no dangling references.
  - All foreign key constraints are enforced.
  - An example of a data model without Referential Integrity: HTML

Foreign Keys in SQL

- Only students listed in the Student relation should be allowed to have grades for courses that are listed in the Course relation. This makes business sense because you have to be a registered student to take courses.

```
CREATE TABLE Grade
  (sid INTEGER, dept CHAR(4), course# CHAR(3), mark INTEGER,
   PRIMARY KEY (sid, dept, course#),
   FOREIGN KEY (sid) REFERENCES Student (sid),
   FOREIGN KEY (dept, course#) REFERENCES Course (dept, cnum) )
```

Primary key in Student

Primary key in Course; columns don't need to have the same name

Grade

sid	dept	course#	mark
53666	CPSC	101	80
53666	RELG	100	45
53650	MATH	200	null
53666	HIST	201	60

Student

sid	name	address	Phone	major
53666	G. Jones	….	…	…
53688	J. Smith	….	…	…
53650	G. Smith	….	…	…

Self Referencing Relations

Goal: Have managerID be a foreign key reference for the same table: Employee.

id	sin	name	managerID
1	1000	Jane	Null
2	1001	Jack	1

- Can a foreign key be null?
- For referential integrity to hold in a relational database, any field in a table that is declared a foreign key should contain either a null value, or only values from a parent table's key.

Clicker Question

Consider the table definition: CREATE TABLE Employee (

Numbers denote lines only →

```
(1) id          INTEGER,
(2) sin         INTEGER,
(3) name        CHAR(20),
(4) managerID   INTEGER );
```

Goal: Have managerID be an FK reference for the same table Employee.

Which of the following is **not** legal? (does not have to achieve all goals)

A. Add FOREIGN KEY (managerID) REFERENCES Employee (id) just before the) on line (4).

B. Add PRIMARY KEY just before the comma on lines (1) and (2), and add REFERENCES Employee (id) just before the) on line (4).

C. Add PRIMARY KEY just before the comma on line (1), add UNIQUE just before the comma on line (2), and add FOREIGN KEY REFERENCES Employee (sin) just before the) on line (4).

D. All of the above are legal

E. None of the above are legal

Enforcing Referential Integrity

- sid in Grade is a foreign key that references Student
- What should be done if a Grade tuple with a non-existent student ID is inserted? (*Reject it!*)
- What should be done if a **Student tuple** is deleted? Should we:
  - Delete all Grade tuples that refer to it?
  - Disallow deletion of this particular Student tuple?
  - Set sid in all Grade tuples that refer to it, to null (the special value denoting *unknown* or *not applicable*)?
    - Note: This is a problem if sid is part of Grade's primary key.
  - Set sid in all Grade tuples that refer to it, to a *default* sid?
- Similarly, if the PK of a Student tuple is updated

Referential Integrity in SQL/92

- SQL/92 supports all 4 options on deletes and updates:
  - The default is **NO ACTION** (i.e., delete/update is rejected).
  - **CASCADE** (also deletes/updates all tuples that refer to the deleted/updated tuple)
  - **SET NULL / SET DEFAULT** (the referencing tuple value is set to the default foreign key value)

```
CREATE TABLE Grade
   (sid CHAR(8),  dept  CHAR(4),
    course# CHAR(3),  mark INTEGER,
    PRIMARY KEY  (sid, dept, course#),
    FOREIGN KEY (sid)
        REFERENCES Student
           ON DELETE CASCADE
           ON UPDATE CASCADE
    FOREIGN KEY (dept, course#)
        REFERENCES Course (dept, cnum)
           ON DELETE SET DEFAULT
           ON UPDATE CASCADE );
```

Clicker Question

Consider the following table definition.

```
CREATE TABLE  BMW  ( bid  INTEGER, sid INTEGER,  …
                    PRIMARY KEY (bid),
                    FOREIGN KEY (sid) REFERENCES STUDENT
                                ON DELETE CASCADE);
```

Suppose bid = 1000 and sid = 5678 for a row in Table BMW.

A. If the row for sid value 5678 in STUDENT is deleted, then the row with bid = 1000 in BMW is automatically deleted.

B. If a row with sid value 5678 in BMW is deleted, then the row with sid = 5678 in STUDENTS is automatically deleted.

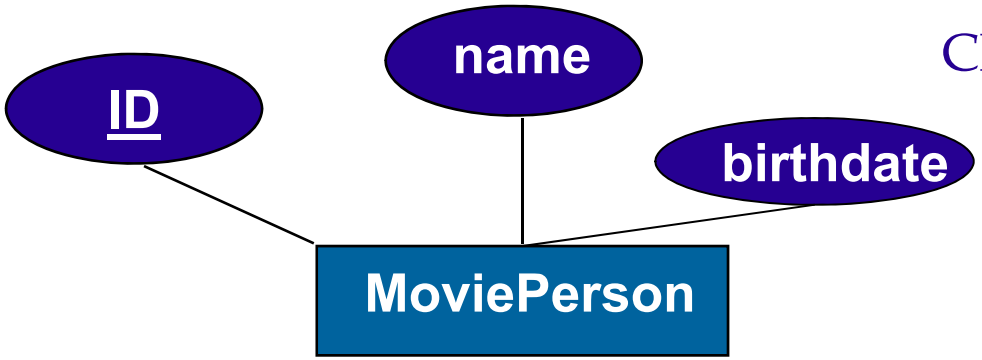
C. Both of the above

Where Do ICs Come From?

- ICs are based upon the real-world semantics being described (in the database relations).
- We *can* check a database instance to verify an IC, but we *cannot* tell the ICs by looking at the instance.
  - For example, even if all student names differ, we cannot assume that name is a key.
  - An IC is a statement about *all possible* instances.
- All constraints must be identified during the conceptual design.
- Some constraints can be explicitly specified in the conceptual model.
  - Key and foreign key ICs can be shown on ER diagrams.
- Others are written in a more general language.

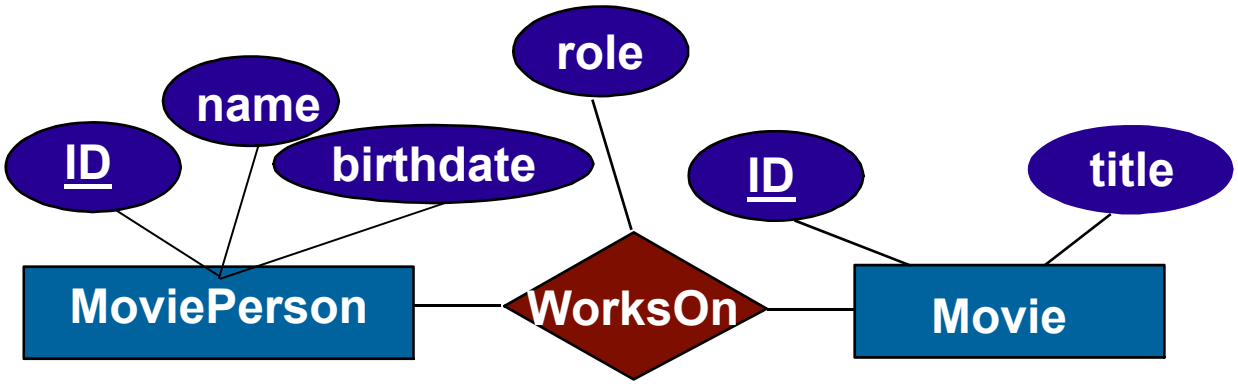
Logical DB Design: ER to Relational

- Each entity set is mapped to a relation.
  - Entity attributes become table attributes.
  - Entity keys become table keys.




```
CREATE TABLE MoviePerson
  (ID          CHAR(11),
   name        CHAR(20),
   birthdate   DATE,
   PRIMARY KEY (ID) )
```

Relationship Sets to Tables

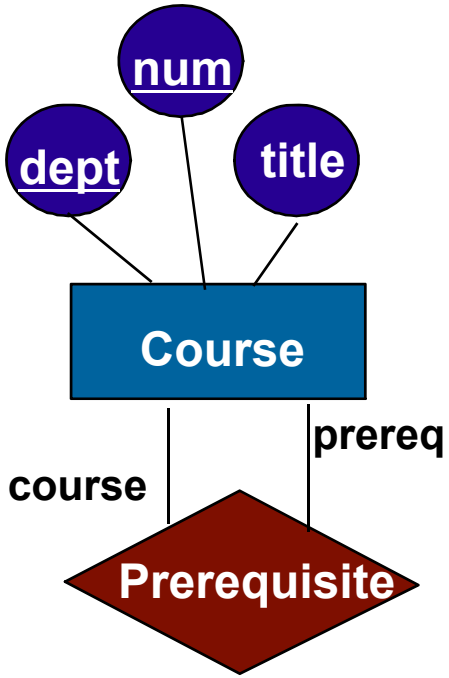


- A relationship set id is mapped to a single relation (table).
- Simple case: relationship has no constraints (i.e., many-to-many)
- In this case, attributes of the table must include:
  - Keys for each participating entity set as foreign keys
    - This is a key for the relation.
  - All descriptive attributes

```
CREATE TABLE WorksOn (
  pid     CHAR(11),
  mid     INTEGER,
  role    CHAR(20),
  PRIMARY KEY (pid, mid),
  FOREIGN KEY (pid)
      REFERENCES MoviePerson,
  FOREIGN KEY (mid)
      REFERENCES Movie)
```

pid and mid CANNOT be null and were renamed

Relationship Sets to Tables (cont.)

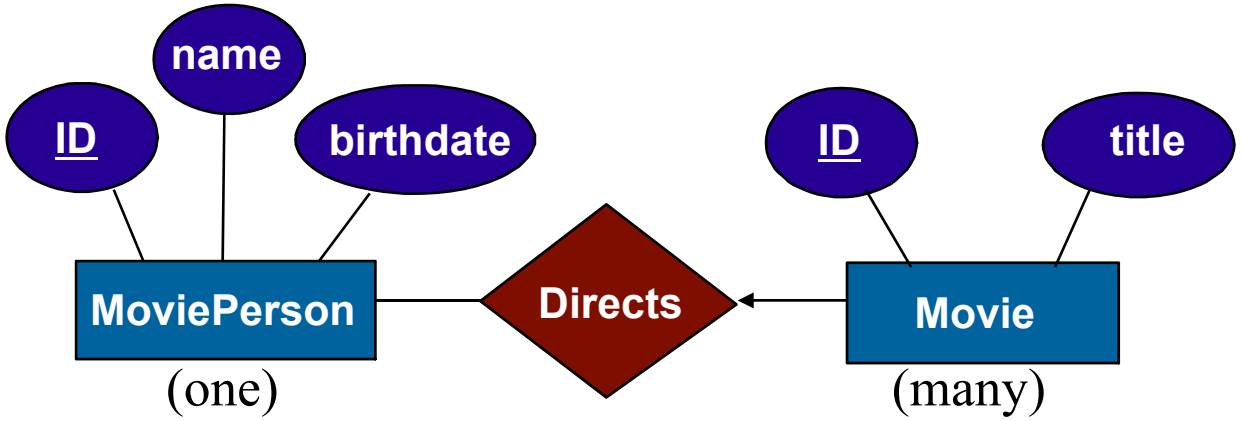


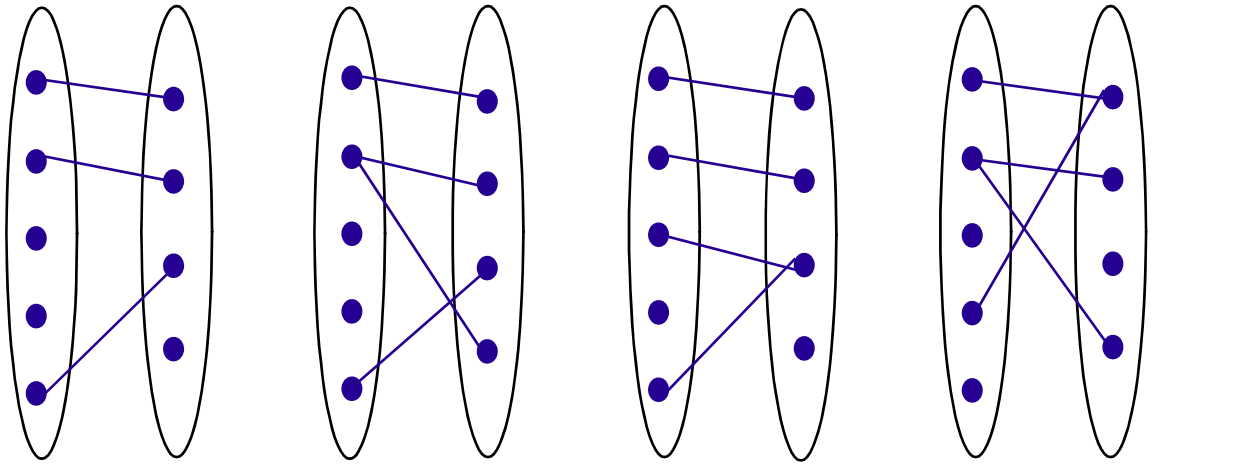
- In some cases, we need to use the roles:

```
CREATE TABLE Prerequisite (
    course_dept     CHAR(4),
    course_num      CHAR(3),
    prereq_dept     CHAR(4),
    prereq_num      CHAR(3),
    PRIMARY KEY (course_dept, course_num,
                 prereq_dept, prereq_num),
    FOREIGN KEY (course_dept, course_num)
        REFERENCES Course(dept, num),
    FOREIGN KEY (prereq_dept, prereq_num)
        REFERENCES Course(dept, num) )
```

Review: Key Constraints

- Each movie has at most one director, according to the key constraint on Directs.

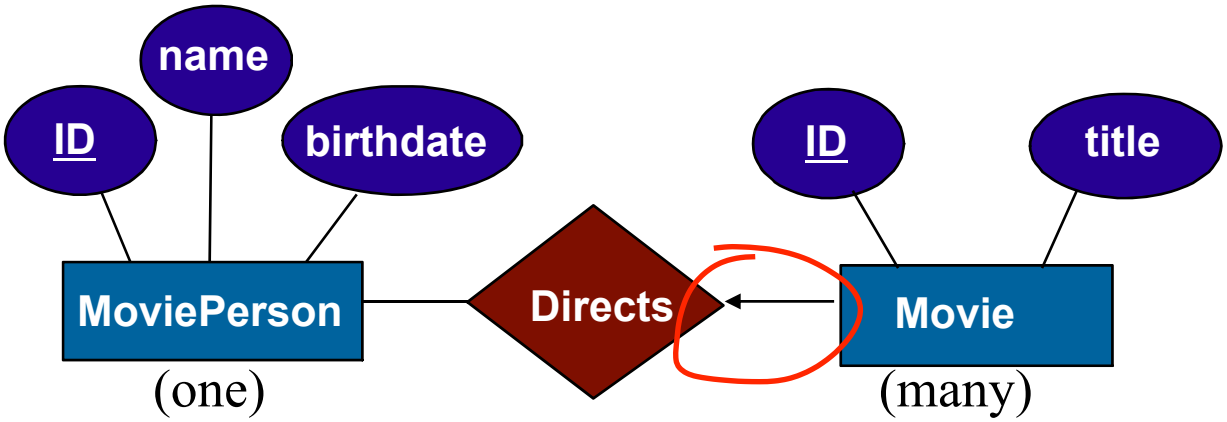




1-to-1 | 1-to Many | Many-to-1 | Many-to-Many

Translation to relational model?

Relationship Sets with Key Constraints



- Each movie has at most one director, according to the key constraint on Directs.
- How can we take advantage of this?

Translating ER Diagrams with Key Constraints

- Method 1 (unsatisfactory):
  - Create a separate table for Directs:
  - Note that mid is the key now!
  - Create separate tables for MoviePerson and Movie.

```
CREATE TABLE  Directs (
  pid      CHAR(11),
  mid      INTEGER,
  PRIMARY KEY  (mid),
  FOREIGN KEY (pid) REFERENCES MoviePerson,
  FOREIGN KEY (mid) REFERENCES Movie )
```

unsatisfactory

Eventually 3 tables

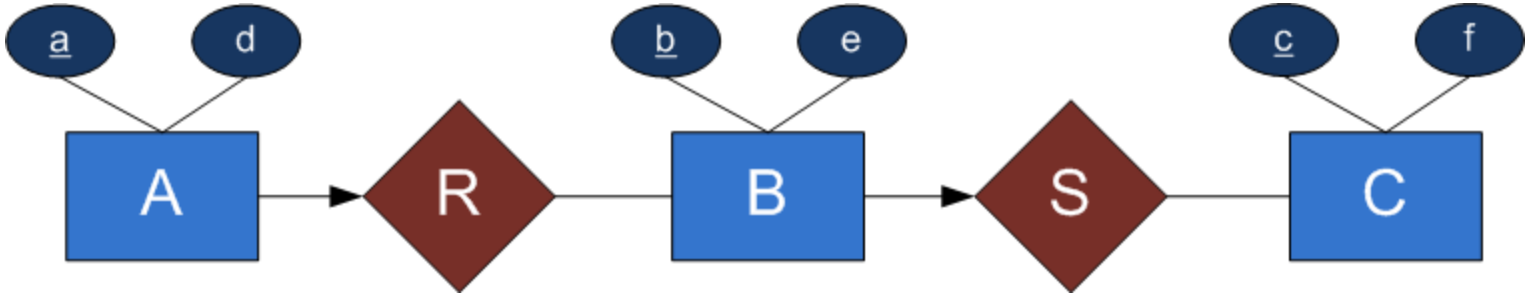
- Method 2 (better)
  - Since each movie has a unique director, we can **combine Directs and Movie into one table.**
  - Create another table for MoviePerson.

```
CREATE TABLE  Directs_Movie (
  mid      INTEGER,
  title    CHAR(20),
  pid      CHAR(11),
  PRIMARY KEY  (mid),
  FOREIGN KEY (pid) REFERENCES MoviePerson
              ON DELETE SET NULL
              ON UPDATE CASCADE )
```

pid may be null

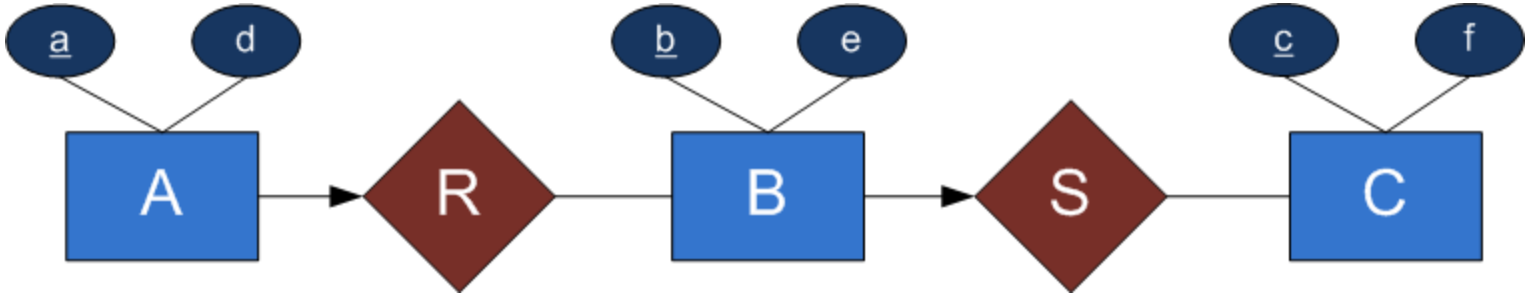
Oracle does not support "on update".

Exercise, Followed by a Clicker Question



- Translate the ER diagram to a relational schema.
- Then, underline the key attributes, and make the FKs bold.

Clicker Question about Your Solution



Translate the ER diagram to relational ...

Which of the following appears in your relational schema?

A. AR(a, **b**, d) **b shouldn't be part of the key**

B. BS(b, **c**, e)

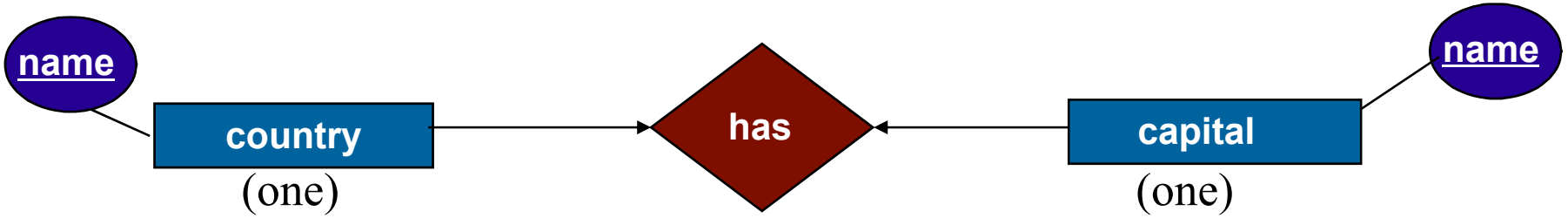
C. S(b, **c**)

D. All of these

E. None of these

C(c(undrsore), f)

Clicker Question: Relationship Sets with Key Constraints (One-to-One Case)



Which schema below is a reasonable translation from ER to relations, assuming that a country can only have one capital city?

A. Country(coName, caName)
B. Country(name), Capital(name)
C. Capital (caName, coName)
D. Both A and C
E. All of A, B, and C