CPSC 368
Databases in Data Science

The Relational Model

Textbook Reference
Database Management Systems: 3.1 - 3.5

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
- In our last very exciting episode, we showed how to use ER diagrams to design the *conceptual schema*
- But the conceptual schema can only get us so far; we need to store data!
- Now we'll learn to use a *logical schema* to actually store the data. We'll be using the *relational model*.

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 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 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.

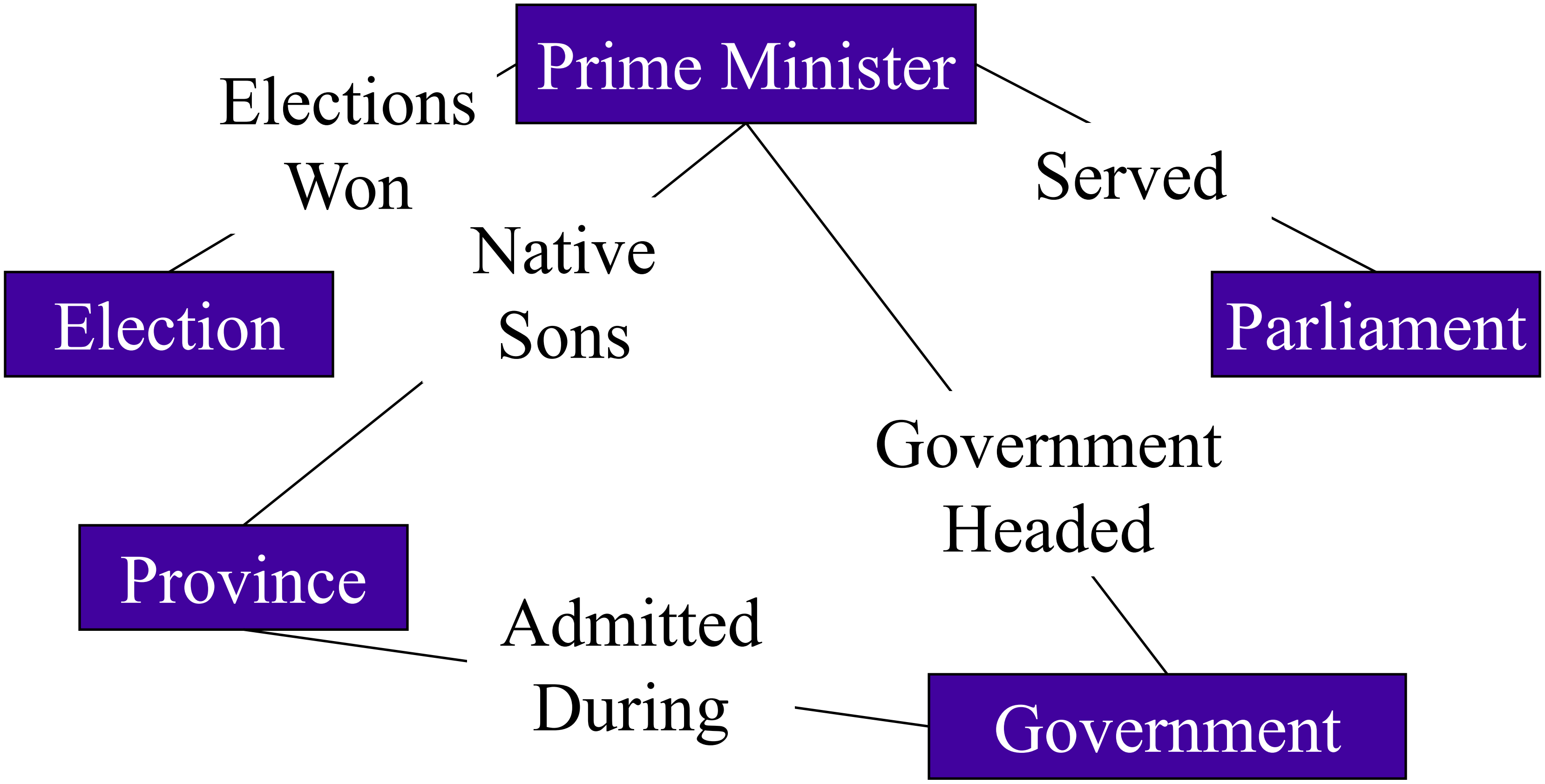
What do we want out of our logical schema representation?

- Ability to store data – w/o worrying about blocks on disk
- Ability to query data easily
- A representation that is easy to understand
- A representation that we can easily adapt from conceptual schema
- Separate from application programming language

How did we get the relational model?

- Prior to the relational model, there were two main contenders
  - Network databases
  - Hierarchical databases
- Network databases had a complex data model
- Hierarchical databases integrated the application in the data model

Example Hierarchical Model



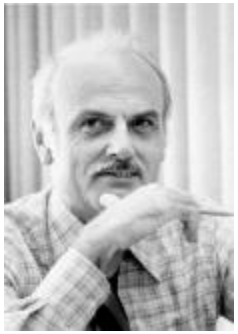
Example IMS (Hierarchical) query: Print the names of all the provinces admitted during a Liberal Government

```
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 BIRARY(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;
```

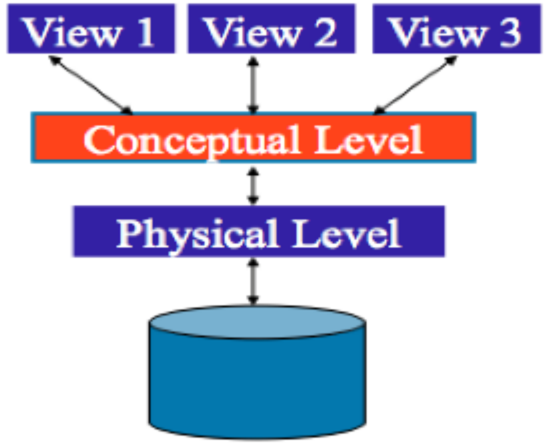
Relational model to the rescue!



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

Key points of the relational model

- Exceedingly simple to understand – main abstraction is represented as a table
- **Physical Data Independence** –ability to modify physical schema w/o changing logical schema
- **Logical Data Independence** – done with views
    - Ability to change the conceptual schema without changing applications



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.
    #Rows = cardinality
    #Columns = arity / degree
- ***Relational Database Schema:*** collection of schemas in the database
- ***Database Instance:*** a collection of instances of its relations

Example of a Relation Instance

relation name

attribute, column name

tuple, row, record

domain value

Student

sid	name	address	phone	major
99111120	K. Jones	1234 W. 12^{th} Ave., Van	889-4444	CPSC
92001200	S. Selvarajah	2020 E. 18^{th} St., Van	409-2222	MATH
94001020	A. Alberty	2020 E. 18^{th} St., Van	222-2222	FREN
94001150	J. Wang	null	null	null

- 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 $(a_1, a_2,\ldots,a_n)$ 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.
- If $A_1,\ldots, A_n$ are attributes with domains $D_1,\ldots D_n$, then
  $(A_1{:}D_1, \ldots, A_n{:}D_n)$
  is a ***relation schema*** that defines a relation type – sometimes we leave off the domains

Example of a formal definition

Student

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

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

Or without the domains:

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.

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

Which of the following is NOT a true statement about R?

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: simple, powerful *querying* of data.

- Queries can be written intuitively; DBMS is responsible for efficient evaluation.
  - Precise semantics for relational queries.
  - Allows optimizer to extensively re-order operations, while ensuring that the answer does not change.

The SQL Query Language

- SQL was **NOT** the first relational query language
- 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 SQL

Students

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

- Find the id’s, names and phones of all CPSC students:

```
SELECT  sid, name, phone
FROM    Students
WHERE   major=“CPSC”
```

sid	name	phone
99111120	K. Jones	889-4444

- To select whole rows , replace “SELECT sid, name, phone ” with “SELECT * ”

Simple, eh?

- We'll see more about how to query (**data manipulation language**) in Chapter 5.

- But you can't query without having a place to store your data, so back to how to create relations (**data definition language**)

Creating Relations in SQL/DDL

- The statement on the right creates the Student relation
  - the type (domain) of each attribute is specified and enforced when 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 right creates Grade information about courses that a student takes

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

Destroying and Altering Relations

DROP TABLE Student

- Destroys the relation Student. Schema information *and* tuples are deleted.

ALTER TABLE Student
ADD COLUMN gpa REAL;

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

Adding and Deleting Tuples

- Can insert a single tuple using:

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

- 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)

"Integrity is doing the right thing, even when no one is watching"
- CS Lewis

- **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
  - Avoids data entry errors, too!
- The types of IC's depend on the data model.
  - What did we have for ER diagrams?
  - Next up: constraints for relational databases

Quick Detour: Keys

Student

sid	CWL	SIN	name	major	age	…
1	bpuff1	123	Blossom	Music	18	…
2	bpuff2	234	Buttercup	Physics	18	…
3	bpuff3	456	Bubbles	Education	18	…

Candidate keys:

- sid
- CWL
- SIN

Quick Detour: Keys

Student

sid	CWL	SIN	name	major	age	…
1	bpuff1	123	Blossom	Music	18	…
2	bpuff2	234	Buttercup	Physics	18	…
3	bpuff3	456	Bubbles	Education	18	…
4	bBPuff	222	Blossom	Education	18	…

In this strange school, every student within the same major must have a unique name.

E.g., There is a student called Blossom in music so the music major can never admit another student named Blossom.

Quick Detour: Keys Clicker Question

Student

sid	CWL	SIN	name	major	age	…
1	bpuff1	123	Blossom	Music	18	…
2	bpuff2	234	Buttercup	Physics	18	…
3	bpuff3	456	Bubbles	Education	18	…
4	bBPuff	222	Blossom	Education	18	…

In this instance, could {major, name} possibly be a candidate key?

A. Yes
B. No

1. No distinct tuples can have the same values for all attributes in the key, and
2. No proper subset of the potential key is itself a key (according to (1)).

Quick Detour: Keys (primary keys)

Student

sid	CWL	SIN	name	major	age	…
1	bpuff1	123	Blossom	Music	18	…
2	bpuff2	234	Buttercup	Physics	18	…
3	bpuff3	456	Bubbles	Education	18	…
4	bBPuff	222	Blossom	Education	18	…

<u>Candidate keys</u>:

- sid
- CWL
- SIN
- {name, major}

Pick a candidate key to be your primary key

Explain this clicker question

If A is a one of three candidate keys of relation R, does that automatically mean A is the primary key?

A. Yes
B. No
C. It depends

Why?

Keys Constraints (for Relations)

- Similar to those for entity sets in the ER model
- One or more attributes in a relation form a ***key*** (or ***candidate key***) for a relation, where S is the set of all attributes in the key, if:
  1. No distinct tuples can have the same values for all attributes in the key, 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.)
- One of the possible keys is chosen (by the DBA) to be the ***primary key*** (PK).

```
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
  - values for primary key must be unique
  - a primary key attribute cannot be *null*
- Other keys are specified using the **UNIQUE** constraint
  - values for a group of attributes must be unique (if they are not null)
  - these attributes can be *null*
- Key constraints are checked when
  - new values are inserted
  - values are modified

Keys Constraints in SQL (cont')

(Ex.1- Normal) "For a given student and course, there is a single grade."

vs.

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

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

```
CREATE TABLE Grade2
  (sid        INTEGER,
   dept       CHAR(4),
   course#    CHAR(3),
   mark       CHAR(2),
   PRIMARY KEY (sid,dept,course#),
   UNIQUE (dept,course#,mark) )
```

Keys Constraints in SQL (cont')

For single attribute keys, can also be declared on the same line as the attribute.

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

Foreign Keys Constraints

- ***Foreign key*** : Set of attributes in one relation used to 'reference' a tuple in another relation.
  - **Must correspond to the primary key of the other relation.**
  - Like a 'logical pointer'.
- E.g.: Grade(*sid, dept, course#, grade*)
  - *sid* is a foreign key referring to Student:
  - (*dept, course#*) is a foreign key referring to Course
- *Referential integrity*: All foreign keys reference existing entities.
  - i.e. there are no dangling references
  - all foreign key constraints are enforced