INFO 210: Database Management Systems

Topic 2:

The Relational Model

supplementary material:
"A First Course in Database Systems" Ch. 2 (ignore semi-structured)
Sec. 7.1-7.3

Topic 1 Review

Summary

- Take-home message: Databases are cool
- Differences between databases and file systems
- The relational model
- The ER model
- Example of a database schema
- Examples of SQL queries
- A key concept: data independence
- A key concept: SQL is declarative

Useful abbreviations

- DB - database
- DBMS - database management system
- RDBMS - relational database management system
- DBA - database administrator
- SQL - structured query language
  - DDL - data definition language
  - DML - data manipulation language
- ER - entity-relationship
- ERM - entity-relationship model

Dropping relations

```
drop table Enrollment;
drop table Courses;
drop table Departments;
drop table Students;
```

why dropped in this particular order?

is this the only possible order?

Dropping relations (solution)

Recall the create table statements for these tables

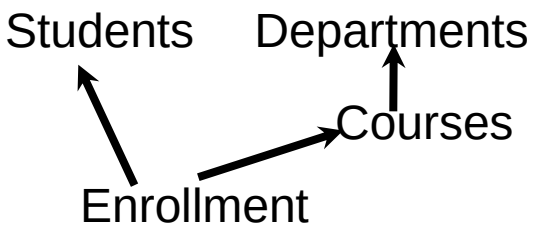
```
create table Students (
  sid number primary key,
  name varchar(128) not null,
  gpa number
);
```

```
create table Departments (
  did number primary key,
  name varchar(128) not null
);
```

```
create table Courses (
  cid number,
  did number,
  name varchar(128) unique,
  credits number default 3,
  primary key (cid, did),
  foreign key (did)
        references Departments(did)
);
```

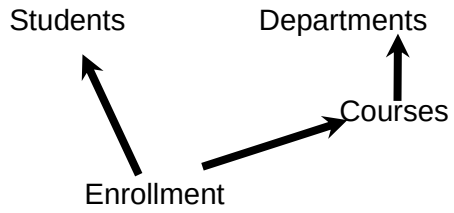
```
create table Enrollment (
  did number,
  cid number,
  sid number,
  term varchar(32),
  grade char(2),
  primary key (cid, did, sid),
  foreign key (cid, did)
        references Courses(cid, did),
  foreign key (sid)
        references Students(sid)
);
```

Note that foreign keys introduce dependencies between these relations.



Dropping relations (solution)

Note that foreign keys introduce dependencies between these relations.



Consider the directed acyclic graph (DAG) of dependencies above.

We cannot drop a relation that has incoming edges in the DAG.

Thus, the following orders are allowed:

```
drop table Enrollment;
drop table Students;
drop table Courses;
drop table Departments;
```

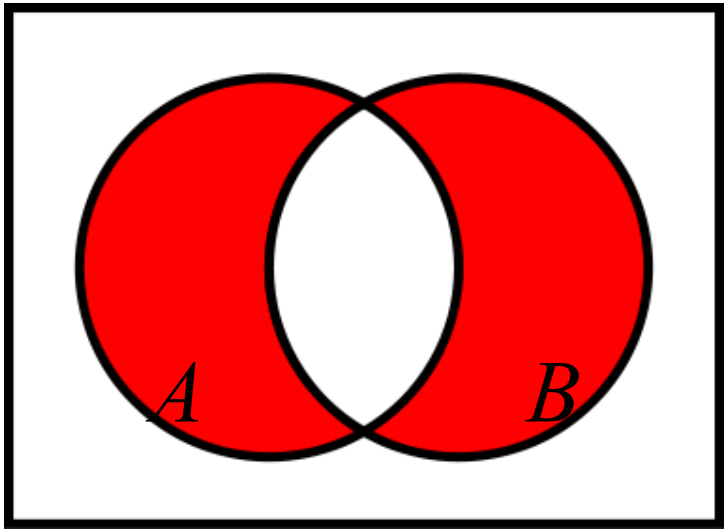
```
drop table Enrollment;
drop table Courses;
drop table Departments;
drop table Students;
```

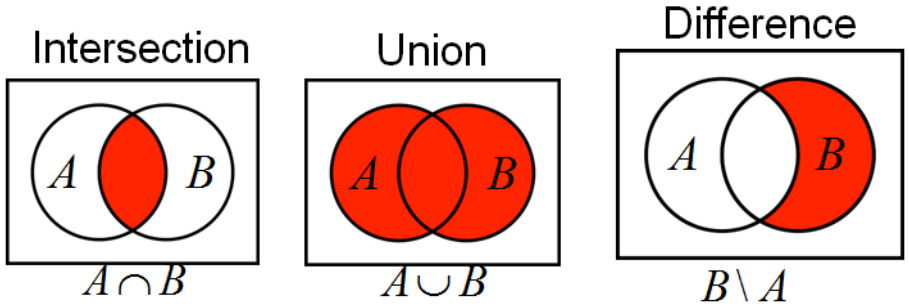
```
drop table Enrollment;
drop table Courses;
drop table Students;
drop table Departments;
```

These correspond to *topological orders* of the nodes in the DAG above.

Set operations

How can you express *symmetric difference* using other set operations?





$$(A \cup B) \setminus (A \cap B)$$

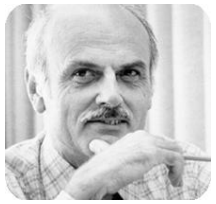
$$(A \cup B) - (A \cap B)$$

Topic 2 outline

- Part 1: Relational model basics
- Part 2: Creating and modifying relations
- Part 3: Foreign key constraints
- Part 4: Summary

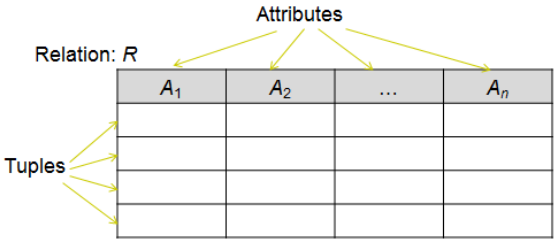
Part 1: relational model basics

- Introduced by Edgar F. Codd in 1970 (Turing award 1981)



- At the heart of relational database systems
  - the basic abstraction is a *relation* (a ***table***)
  - *tuples* are stored in rows
  - *attributes* of tuples are stored in columns
  - conceptually, a relation is a set of tuples

- Relation ⟺ Table
- Attribute ⟺ Column
- Tuple ⟺ Row



- Why this model?
  - Simple yet powerful
  - Great for processing very large data sets in bulk

Relational model: basics

Students

sid	name	gpa
1234	Joe	3.2
5678	Ann	4.0
2468	Mike	3.5

- Relation: a set or tuples - order doesn't matter, all tuples are distinct from each other
- Attribute: a column in a relation (e.g., name)
- Domain: data type of an attribute (e.g., name:string)
- Tuple: a row in a relation, e.g., (1234, Joe, 3.2)
- Schema: description of a relation in terms of relation name, attribute names, attribute datatypes

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

Schema vs. instances

schema

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

a schema describes
all valid instances
of a relation

instance 1

Students

sid	name	gpa
1234	Joe	3.2
5678	Ann	4.0

instance 2

Students

sid	name	gpa
1	Joe	3.2
2	Ann	
3	Mike	3.5

instance 3

Students

sid	name	gpa

Equivalent representations of a relation

- Relation schemas are sets of attributes
  - The order in which attributes appear does not matter
  - Each attribute appears (at most) once
- Relation instances are sets of tuples
  - The order in which tuples appear does not matter
  - Each tuple appear in an instance at most once

These are all equivalent representations!

Students

sid	name	**gpa**
1234	Joe	3.2
5678	Ann	4.0

Students

sid	name	**gpa**
5678	Ann	4.0
1234	Joe	3.2

Students

sid	gpa	**name**
5678	4.0	Ann
1234	3.2	Joe

Integrity constraints

- Ensure that data adheres to the rules of the application
- Specified when schema is defined
- Checked and enforced by the DBMS when relations are modified (tuples added / removed / updated)
- Must hold on every valid instance of the database

1. *Domain constraints* - specify valid data types for each attribute, e.g., Students (*sid*: integer, *name*: string, *gpa*: real)
2. *Key constraints* - define a unique identifier for each tuple
3. *Referential integrity constraints* - specify links between tuples

Key constraints

Definition: A set of attributes is a *candidate key* for a relation if:

1. No two distinct tuples can have the same values for all key attributes (candidate key identifies a tuple)
2. This is not true for any subset of the key attributes (candidate key is minimal)

- If #2 is false (the set of attributes is not minimal) we have a *superkey*

- If there are 2 or more candidate keys
  - one of them is chosen (by the DBA) to be the *primary key*
  - other candidate keys are marked as such using the UNIQUE keyword

Keys: an example

Students (*sid*: integer, *login*: string, *name*: string, *dob*: date)

1. What could be the candidate keys?
2. For each candidate key, what are all the super keys?

Option 1 candidate key = {*sid*}

super keys = { {*sid login*}, {*sid name*}, {*sid dob*}, {*sid login name*}, {*sid login dob*}, {*sid name dob*}, {*sid login name dob*} }

Option 2 candidate key = {*login*}

super keys = { {*login sid*}, {*login name*}, {*login dob*}, {*login sid name*}, {*login sid dob*}, {*login name dob*}, {*login sid name dob*} }

Option 3 candidate key = {*name dob*}

super keys = { {*name dob sid*}, {*name dob login*}, {*name dob sid login*}}

Keys: more examples

Movies (*title*: string, *genre*: string, *year*: integer, *length*: integer)

1. What could be the candidate keys?
2. For each candidate key, what are all the super keys?

Movie_Stars (*name*: string, *dob*: date, *active_start*: date, *active_end*: date)

1.What could be the candidate keys?

2.For each candidate key, what are all the super keys?

Part 2: Creating and modifying relations

1. Translate domain constraints to vendor-specific data types (Oracle)
2. Implement key constraints

Departments (*did*: integer, *name*: string)

```
create table Departments (
  did number primary key,
  name varchar(128) not null
);
```

Every relation must have a primary key!

This relation has 1 candidate key, it becomes the primary key.

Oracle datatypes

number - can store any kind of a number

char(x) - stores a string of length exactly x

varchar(x) - stores a string of length *at most* x, same as varchar2(x)

date - stores a date and time (up to a second)

timestamp - stores a date and time (up to a fraction of a second)

This is a very basic list, for complete information see:

https://docs.oracle.com/cd/B28359_01/server.111/b28318/datatype.htm#CNCPT012

Specifying required attributes

To specify that each department must have a name, we use the construct not null

```
create table Departments (
  did number primary key,
  name varchar(128) not null
);
```

Departments

did	name
1	CS
2	Italian

Departments

did	name
1	CS
2	

Departments

did	name
1	CS
2	Italian
3	Math
1	English

Which of these instances are illegal and why?

Creating relations

1. Translate domain constraints to vendor-specific data types (Oracle)
2. Implement key constraints

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

```
create table Students (
   sid   number       primary key,
   login varchar(128) unique,
   name  varchar(128),
   dob   date,
   gpa   number,
   unique (name, dob)
);
```

Every relation must have exactly one primary key!

This relation has 3 candidate keys

1. *sid* becomes the primary key
2. *login* is designated unique
3. the combination of *name* and *dob* is designated unique

Primary keys may be composite

1. Translate domain constraints to vendor-specific data types (Oracle)
2. Implement key constraints

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

```
create table Students (
   sid   number       unique,
   login varchar(128) unique,
   name  varchar(128),
   dob   date,
   gpa   number,
   primary key (name, dob)
);
```

Every relation must have exactly one primary key!

This relation has 3 candidate keys

1. *sid* bis designated unique
2. *login* is designated unique
3. the combination of *name* and *dob* becomes the primary key

How do we choose a primary key?

- If there is only one candidate key - that's our primary key, no choice

- If multiple candidate keys are available, chose one that
  - does not contain any attributes that may be null
  - is simple (as opposed to composite), or made up of fewer columns if composite
  - is compact (number better than varchar(128))

- It is sometimes recommended to add a simple, compact column to the relation schema, to use as a primary key, e.g., add a "synthetic" id column
- Another important consideration: choose a candidate key that is unlikely to change

Primary key vs. UNIQUE

```
create table Employees (
   id    number       primary key,
   login varchar(128) unique,
   name  varchar(128) not null
);
```

- Primary key
  - No two tuples can agree on values for (all attributes) in a primary key
  - All attributes that make part of a primary key must have assigned values (i.e., they may not be null)

- UNIQUE
  - No two tuples can agree on values for unique attributes (or combinations of attributes designated unique)
  - However, unique attributes may be null

Primary key vs. UNIQUE

```
create table Employees (
   id    number       primary key,
   login varchar(128) unique,
   name  varchar(128) not null
);
```

Instance 1

id	login	name
1	jim	Jim Morrison
2	amy	Amy Winehouse
3		Ethan Coen
4	raj	Raj Kapoor

Instance 2

id	login	name
1		Jim Morrison
2	amy	Amy Winehouse
3		Ethan Coen
4	raj	Raj Kapoor

Instance 3

id	login	name
1	raj	Raj Kapoor
2	amy	Amy Winehouse
3		Ethan Coen
4	raj	Raj Kapoor

Instance 4

id	login	name
1	jim	Jim Morrison
2	amy	
3		Ethan Coen
4	raj	Raj Kapoor

Which of these instances are illegal and why?

Creating relations with SQL

Consider relation schemas and business rules below.
Write create table statements that encode these relation schemas.
Give an example of a valid relation instance.

Famous_Scientist (*name*: string, *dob*: date, *field*: string, *employer*: string)

No two scientists have the same *name*.
No two scientists have the same combination of *dob* and *field* of study.
Each scientist has a *name*, a *dob* and a *field* of study.
Not all scientists are *employed*.

Space_Ship (*name*: string, *seq_num:* number, *country*: string, *captain:* string, *flight*: date)

No two space ships have the same name and sequence number (*seq_num*).
No two space ships have the same combination of *captain* and *flight* date.
Each space ship has a name, a seq_num, and a country.
Not all space ships have been flown or have a captain.

Modifying relation schemas with SQL

Deleting a relation (removes both the schema and the data)

```
drop table Students;
```

Changing the schema of a relation

```
alter table Students add  class char(4);
alter table Students add  class char(4) default 2017;
alter table Students drop column gpa;
```

Where do business rules come from?

- Business rules are given: by the client, but the application designer, by your boss
- A relation schema encodes business rules
- We can never-ever-ever deduce business rules by looking at the relation instance!
  - We can sometimes know which rules do not hold, but we cannot be sure which rules hold

Employee

id	login	name
1	jim	Jim Morrison
2	amy	Amy Winehouse
3	amy	Amy Pohler
4	raj	Raj Kapoor

1. Which column is not a candidate key?
2. Which column(s) may be a candidate key?
3. Give 2 create table statements for which this instance is valid.

Part 3: Foreign key constraints

- A foreign key defines a (directed) link between tuples in different relations
- Foreign keys enforce *referential integrity*: a requirement that a value in an attribute or attributes of a tuple in one relation must also appear as a value in another relation
- Example
  - *Students*(sid), *Courses*(cid), *Enrollment*(sid, cid, grade)
  - *Enrollment*.sid must refer to an existing student, i.e., must match *Students*.sid for some tuple in *Students*
  - *Enrollment*.cid must refer to an existing course, i.e., must match *Courses*.cid for some tuple in *Courses*

Foreign keys: example

```
create table Students (
   sid number primary key,
   login varchar(128) unique,
   name varchar(128) not null,
   dob date not null,
   gpa number,
   unique (name, dob)
);
```

```
create table Enrollment (
   sid number,
   cid number,
   grade number,
   primary key (sid, cid),
   foreign key (sid) references Students(sid),
   foreign key (cid) references Courses(cid)
);
```

Students

sid	login	name	dob	gpa
123	Jane	Jane S.	1/1/90	4.0
456	Ann	Ann M.	5/25/92	3.8

Enrollment

sid	cid	grade
123	210	4.0
456	210	4.0

Declaring foreign key constraints

- A foreign key in a table must reference (point to) a candidate key in another table.
- That is, the target column(s) are either a primary key or are designated UNIQUE.
- Some relational systems limit this further: a foreign key must point to a primary key. For efficiency reasons, this is usually a better choice.

```
create table Students (
   sid   number       primary key,
   login varchar(128) unique,
   name  varchar(128),
   dob   date,
   gpa   number,
   unique (name, dob)
);
```

candidate keys: sid, login, (name, dob)

```
create table Sibling_Pairs (
   one_thing   number,
   two_thing   number,
   primary key (one_thing, two_thing),
   foreign key (one_thing)
               references Students(sid),
   foreign key (two_thing)
               references Students(sid)
);
```

Declaring foreign key constraints

- A foreign key in a table must reference (point to) a candidate key in another table.
- That is, the target column(s) are either a primary key or are designated UNIQUE.
- Some relational systems limit this further: a foreign key must point to a primary key. For efficiency reasons, this is usually a better choice.

```
create table Students (
   sid   number       primary key,
   login varchar(128) unique,
   name  varchar(128),
   dob   date,
   gpa   number,
   unique (name, dob)
);
```

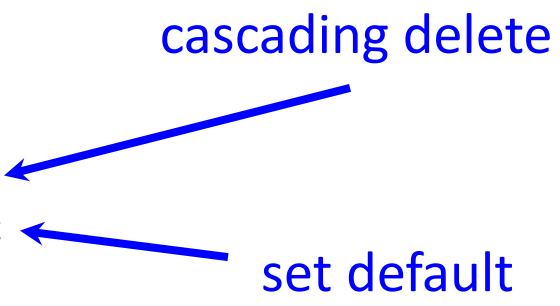
candidate keys: sid, login, (name, dob)

```
create table Sibling_Pairs (
   one_login   varchar(128),
   two_login   varchar(128),
   primary key (one_login, two_login),
   foreign key (one_login)
               references Students(login),
   foreign key (two_login)
               references Students(login)
);
```

Enforcing referential integrity

- Back to *Students* and *Enrollment*; *Enrollment*.sid references *Students*.sid
- What happens if *Enrollment* tuple with a non-existent sid is inserted? (Reject!)
- What should be done if *Students* tuple is deleted, with corresponding *Enrollment* tuples?
  1. Disallow deletion - default behavior
  2. Delete all corresponding *Enrollment* tuples - cascading delete
  3. Set *Enrollment*.sid to a default sid - does not always make sense
  4. Set *Enrollment*.sid to *null* - allowed if not part of primary key, but does not always make sense
- Similar question if *Students.*sid is updated in some tuple

```
create table Enrollment (
   sid number,
   cid number,
   grade number,
   primary key (sid, cid),
   foreign key (sid) references Students(sid) on delete cascade,
   foreign key (cid) references Courses(cid) on delete set default
);
```



Nulls in foreign key columns

```
create table Employees (
   ssn    char(12)     primary key,
   name   varchar(128) not null,
   title  varchar(128)
);
```

```
insert into Employees (ssn, name, title) values ('111-11-1111', 'Ann', 'CEO');
insert into Employees (ssn, name, title) values ('222-22-2222', 'Bob', 'clerk');
```

1. an employee has at most one manager, who is also an employee
2. not every employee has a manager

```
create table Managers (
   emp_ssn    char(12)     primary key,
   mgr_ssn    char(12),
   foreign key (emp_ssn) references Employee(ssn),
   foreign key (mgr_ssn) references Employee(ssn)
);
```

```
insert into Managers (emp_ssn, mgr_ssn) values ('222-22-2222', '111-11-1111');
insert into Managers (emp_ssn, mgr_ssn) values ('111-11-1111', null);
```

Foreign key examples

Consider relation schemas and business rules below. Primary keys are given, underlined in the relation schemas.
Write create table statements that encode these relation schemas, with the right foreign key constraints. Give an example of valid relation instances.

Scientists (*name*: string, *field*: string, *lab*: string)
Labs (*name*: string, *location*: string)

Each scientist works at some lab.
If a lab closes, the corresponding scientist tuples are deleted.

Players (*name*: string, *level*: integer, *country*: string, *favorite_game*: string)
Games (*name*: string, *maker*: string)

All players always play their one favorite game.
If a game is discontinued, its players are made to play Angry Birds.

Foreign key examples

Consider relation schemas and business rules below. Primary keys are given, underlined in the relation schemas.
Write create table statements that encode these relation schemas, with the right foreign key constraints. Give an example of valid relation instances.

Scientists (*name*: string, *field*: string, *lab*: string)
Labs (*name*: string, *location*: string)

Each scientist works at some lab.
If a lab closes, the corresponding scientist tuples are deleted.

```
create table Labs (
                name varchar(128) primary key,
   location varchar(128)
);
create table Scientists (
                name        varchar(128) primary key,
                field       varchar(64),
                lab         varchar(128) not null,
                foreign key (lab) references Labs(name) on delete cascade
);
```

Foreign key examples

Consider relation schemas and business rules below. Primary keys are given, underlined in the relation schemas.
Write create table statements that encode these relation schemas, with the right foreign key constraints. Give an example of valid relation instances.

Players (*name*: string, *level*: integer, *country*: string, *favorite_game*: string)
Games (*name*: string, *maker*: string)

All players always play their one favorite game.
If a game is discontinued, its players are made to play Angry Birds.

```
create table Games (
   name      varchar(128) primary key,
             maker        varchar(128)
);
create table Players (
             name                                  varchar(128) primary key,
             level                                 number,
             country                  varchar(64),
             favorite_game            varchar(128) not null default 'Angry Birds',
             foreign key (favorite_game) references Games(name) on delete set default
);
```

Part 4: Constraints continued

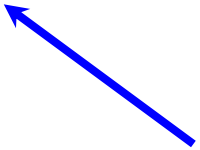
- So far, we talked about enforcing:
  - domain constraints, by specifying data types for attributes
  - not null constraints, by specifying not null for attributes
  - key constraints, by defining primary keys and designating attributes / combinations of attributes as UNIQUE
  - referential integrity constraints, by defining foreign keys

- Another useful type of a constraint is a CHECK constraint

Specifying CHECK constraints

- We already saw an example: not null constraints
- Another example

```
create table Person (
    ssn       char(11) primary key,
    name      varchar(128),
    age       number not null,
    gender    char(1),
    country   varchar(64),
    check (gender in ('M', 'F')),
    check (age > 0),
    check (country in (select name from Country))
);
```



not supported in Oracle 9i

Naming constraints

```
create table Person (
    ssn       char(11) primary key,
    name      varchar(128),
    age       number not null,
    gender    char(1),
    country   varchar(64),
    constraint Gender_Constraint
                  check (gender in ('M', 'F')),
    constraint Age_Constraint
                  check (age > 0),
    constraint Country_Constraint
                  check (country in (select name from Country))
);
```

- Why do we name constraints?
- For readability.
- Also because we can refer to them by name, so as to drop them.

Modifying constraints

```
create table Person (
    ssn      char(11) primary key,
    name     varchar(128),
    age      number not null,
    gender   char(1),
    country  varchar(64),
    constraint Gender_Constraint
                check (gender in ('M', 'F')),
    constraint Age_Constraint
                check (age > 0),
    constraint Country_Constraint
                check (country in (select name from Country))
);
```

```
alter table Person drop constraint Country_Constraint;
```

```
alter table Person add constraint SSN_Constraint
            check (ssn like '%-%-%')
```

Summary

- Relational model: basic definitions
- Enforcing data integrity
  - domain constraints
  - key constraints (candidate key, super key)
  - check constraints
  - foreign key constraints
- Defining relations (with `create table` statements) that implement these constraints
- Next to come: relational algebra