INFO 210: Database Management Systems

Topic 3:

Constraints and the Relational Algebra

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

Topic 2 Review

Summary from last topic

- Relational model: basic definitions
- Enforcing data integrity
  - domain constraints
  - key constraints (candidate key, super key)
  - foreign key constraints
- Defining relations (with create table statements) that implement these constraints

candidate and super Keys:

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?

candidate and super Keys: (solution)

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?

Candidate key 1: {title, year}. This is a possible candidate key if we assume that we never have 2 movies by the same title come out in a particular year.

Super keys: {title, year, genre}, {title, year, length}, {title, year, genre, length}

Candidate key 2: {title, length}. This is a possible candidate key if we assume that, even if 2 movies have the same title, their lengths will differ.

Super keys: {title, length, genre}, {title, length, year}, {title, length, genre, year}

Other candidate keys may be reasonable as well.

Keys: more examples (solution)

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?

Candidate key 1: {name}. This is a possible candidate key if we assume that no two movie stars have the same name.

Super keys: {name, dob}, {name, active_start}, {name, active_end}, {name, dob, active_start}, {name, dob, active_end}, {name, dob, active_start, active_end}

Other candidate keys may be reasonable as well.

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.

Creating relations with SQL (solution)

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*.

```
create table Famous_Scientist (
   name varchar(128) primary key,
   dob date not null,
   field varchar(128) not null,
   employer varchar(128),
   unique (dob, field)
);
```

name	dob	field	employer
Nikola Tesla	06/10/1856	Electrical Engineering	
Niels Bohr	10/7/1885	Physics	University of Copenhagen
Aage Bohr	06/19/1922	Nuclear Physics	Manhattan Project
Marie Curie	11/7/1867	Physics	University of Paris

Creating relations with SQL (solution)

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.

```
create table Space_Ship (
    name varchar(128),
    seq_num number,
    country varchar(128) not null,
    captain varchar(128),
    flight date,
    primary key (name, seq_num),
    unique (captain, flight)
);
```

note that *not null* is not required for name, seq_num, since these make up the primary key, and so are implicitly not null

name	seq_num	country	flight	captain
Vostok	1	USSR	April 12, 1961	Yuri Gagarin
Vostok	6	USSR	June 16, 1963	Valentina Tereshkova
Apollo	11	USA	July 16, 1969	Neil Armstrong
Apollo	13	USA	April 11, 1970	Jim Lovell

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.

Where do business rules come from? (solution)

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?

login is not a candidate key, since the value "amy" appears twice

2. Which column(s) may be a candidate key?

(id), (login, name), (name) are possible candidate keys, since these are unique in the instance. However, we cannot be sure without an exact specification of a business rule.

3. Give 2 create table statements for which this instance is valid.

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

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

note that simply changing the order of columns in the create table statement does not constitute a different create table statement, since relations are sets of columns, and so their row order is immaterial

Topic 3 outline

- Part 1: Constraints Cont’d
- Part 2: Relational algebra
- Part 3: Summary

Part 1: 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

~~foreign key (country) references Country(name)~~

Naming constraints

```
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))
);
```

```
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

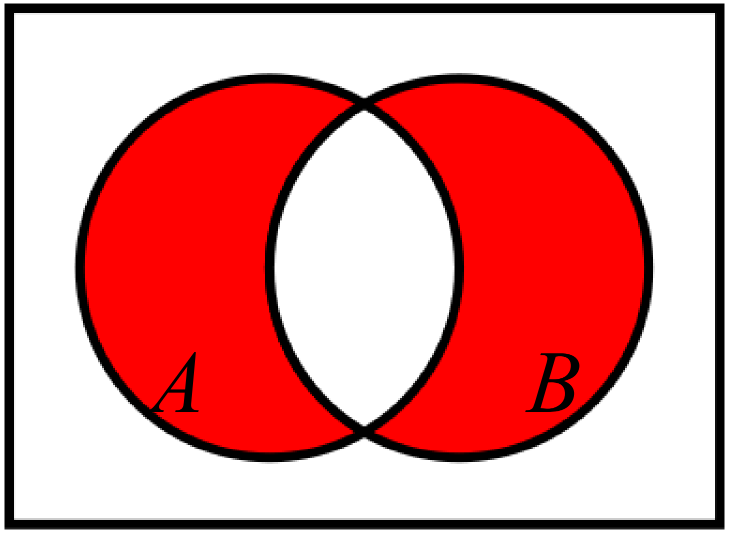
```
create table Person (
   ssn      char(11) primary key,
   name      varchar(128),
   age      number not null,
   gender   char(1),
   country  varchar(64),
   check (gender in ('M', 'F')),
   check (age > 0),
   check (country in (select name from Country))
);
```

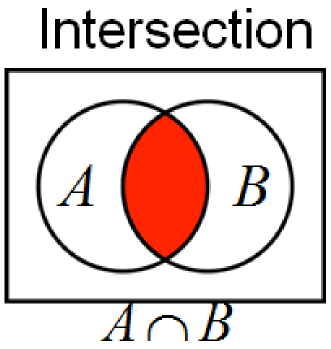
```
create table Person (
   ssn      char(11) primary key,
   name      varchar(128),
   age      number not null,
   gender   char(1),
   country  varchar(64),
   constraint Gender_Constraint
               check (gender in ('M', 'F')),
   constraint Age_Constraint
               check (age > 0),
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

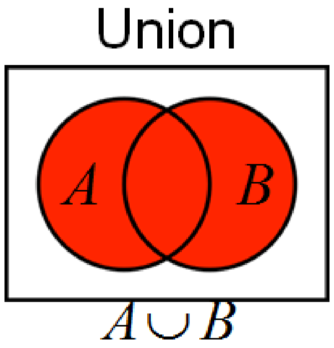
```
alter table Person drop constraint Country_Constraint;

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

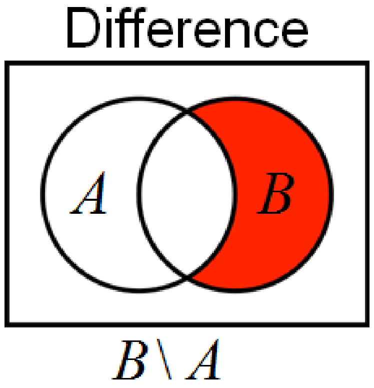
Set operations

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



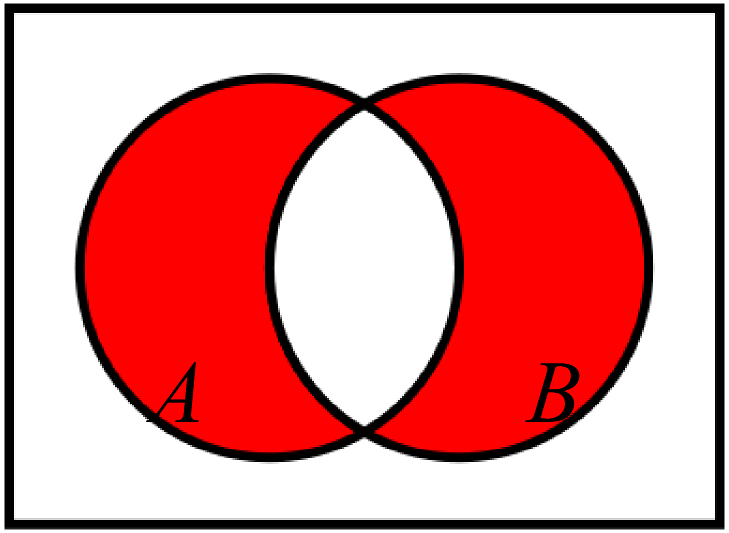


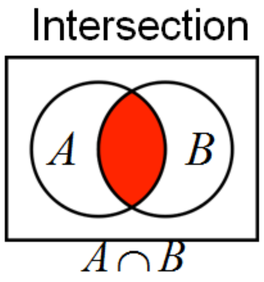




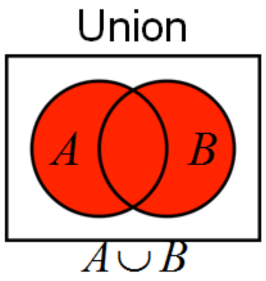
Set operations (solution)

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

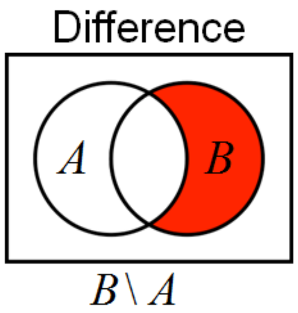




$A \cap B$



$A \cup B$



$B \setminus A$

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

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

$$(A \setminus B) \bigcup (B \setminus A)$$

Part 2: Relational Algebra

- Formally: the data manipulation aspect of the relational model
- Practically: a programming language for relational databases
  - less powerful than a general programming language like Java or C
  - so, (1) easier to learn (for us) and (2) easier to make efficient (for the DBMS)

- SQL (the DML part) implements relational algebra operators
  - understanding relational algebra makes it much easier to understand SQL

- Surprise: relational algebra is based on set operations!

What is an algebra?

- A system consisting of operators and operands
- We are all familiar with the algebra of arithmetic: operators are + − ×, operands are constants, like *42*, or variables, like *x*
- Expressions are made up of operators, operands, optionally grouped by parentheses, e.g., *(x + 3) / (y − 1)*
- What's another algebra we worked with recently?

- Back to relational algebra:
  – operands are variables - stand for relations
  – constants - stand for finite relations (think a particular set of tuples)
  – let's look at operators

Relational algebra operations

- The usual set operations: union ∪ , intersection ∩, set difference \ , but applied to relations (sets of tuples)
- Operations that remove parts of a relation
  - selection removes rows (tuples)
  - projection removes columns (attributes)
- Operations that combine tuples of two relations
  - Cartesian product - pairs tuples in two relations in all possible ways
  - join - selectively pairs tuples from two relations
- A renaming operation changes relation schema, renaming either relation name or names of attributes – don't worry about this for now

Set operations on relations

Definition: Relations *R* and *S are* *union-compatible* if their schemas define attributes with the same (or compatible) domains.

Set operations can only be applied to union-compatible relations.

R

id	name	age
1	Ann	18
2	Jane	22

S

id	name	age
1	Ann	18
3	Mike	21
4	Dave	27

$R \cup S$

id	name	age
1	Ann	18
2	Jane	22
3	Mike	21
4	Dave	27

$R \cap S$

id	name	age
1	Ann	18

R / S

id	name	age
2	Jane	22

S / R

id	name	age
3	Mike	21
4	Dave	27

Note: (1, Ann, 18) appears only once in the result of $R \; U S$

Selection

The selection operator, applied to relation *R*, produces a new relation with a subsets of *R*' s tuples. Tuples in the new relation are those that satisfy some condition *c*.

$\boxed{\sigma_C(R)}$

R

id	name	age	gender
1	Ann	18	F
2	Jane	22	F
3	Mike	21	M
4	Dave	27	M

$\sigma_{age \geq 21}(R)$

id	name	age	gender
2	Jane	22	F
3	Mike	21	M
4	Dave	27	M

$\sigma_{age \geq 21\ AND\ gender='M'}(R)$

id	name	age	gender
3	Mike	21	M
4	Dave	27	M

Note: $\sigma_C(R)$ has at most as many rows as *R*

Projection

The projection operator, applied to relation *R*, produces a new relation with a subsets of *R*' s attributes.

$$\boxed{\pi_{A_1, A_2, \ldots, A_n}(R)}$$

R

id	name	age	gender
1	Ann	18	F
2	Jane	22	F
3	Mike	21	M
4	Dave	27	M

$\pi_{id,\ name}(R)$

id	name
1	Ann
2	Jane
3	Mike
4	Dave

$\pi_{gender}(R)$

gender
F
M

Note: $\pi_{A_1, A_2, \ldots, A_n}(R)$ has at most as many rows as *R*

Cartesian product

The Cartesian product (or cross product) of two relations *R* and *S* is the set of pairs, formed by choosing the first element from *R* and the second element from *S*.

$\boxed{R \times S}$

R

id	name	age
1	Ann	18
2	Jane	22

S

id	name	age
3	Mike	21
4	Dave	27

$R \times S$

R.id	R.name	R.age	S.id	S.name	S.age
1	Ann	18	3	Mike	21
1	Ann	18	4	Dave	27
2	Jane	22	3	Mike	21
2	Jane	22	4	Dave	27

Note: there are exactly $|R| * |S|$ tuples in $R \times S$

Join

The join of two relations *R* and *S* is the set of pairs, formed by choosing the first element from *R* and the second element from *S*, such that the corresponding tuples in *R* and *S* meet some condition *c* .

$R \bowtie_C S$

R

id	name	age
1	Ann	18
2	Jane	22

S

id	name	age
3	Mike	21
4	Dave	27

$R \bowtie_{R.age < S.age} S$

R.id	R.name	R.age	S.id	S.name	S.age
1	Ann	18	3	Mike	21
1	Ann	18	4	Dave	27
~~2~~	~~Jane~~	~~22~~	~~3~~	~~Mike~~	~~21~~
2	Jane	22	4	Dave	27

Note: there are at most $|R| * |S|$ tuples in $R \bowtie_C S$

Join vs. Cartesian product

Conceptually, to compute $R \bowtie_C S$

1. compute a Cartesian product R $\times$ S
2. then compute a selection σ_C (R $\times$ S) using the join condition

$$\boxed{R \bowtie_C S = \sigma_C\,(R \times S)}$$

Natural join

The natural join of two relations *R* and *S* is a simpler kind of a join, where the condition is: pair up tuples from *R* and *S* that agree on the values of the common attributes.

$\boxed{R \bowtie S}$

R

sid	name	gpa
1111	Joe	3.2
2222	Ann	4.0
3333	Mike	3.5

S

sid	did	cid	term	grade
1111	1	210	Fall 2012	A
2222	1	220	Winter 2013	

$R \bowtie S$

R.sid	R.name	R.gpa	S.sid	S.did	S.cid	S.term	S.grade
1111	Joe	3.2	1111	1	210	Fall 2012	A
2222	Ann	4.0	2222	1	220	Winter 2013	

Note: there are at most $|R| * |S|$ tuples in $R \bowtie_C S$

Relational algebra operations (recap)

- The usual set operations: union ∪ , intersection ∩, set difference \ , but applied to relations (sets of tuples)
- Operations that remove parts of a relation
  - selection removes rows (tuples)
  - projection removes columns (attributes)
- Operations that combine tuples of two relations
  - Cartesian product - pairs tuples in two relations in all possible ways
  - join - selectively pairs tuples from two relations

Combining operations into queries

What are the names of students whose GPA is at least 3.5?

Students

sid	name	gpa
1111	Joe	3.2
2222	Ann	4.0
3333	Mike	3.5

1. Select those *Students* tuples that have $gpa \geq 3.5$.
2. Get (project) the value of the *name* column for these tuples.

$\pi_{name} (\sigma_{gpa \geq 3.5} (Students))$

name
Ann
Mike

Combining operations into queries

What are the names of students who got an A in any course?

Students

sid	name	gpa
1111	Joe	3.2
2222	Ann	4.0
3333	Mike	3.5

Enrollment

sid	did	cid	term	grade
1111	1	210	Fall 2012	A
2222	1	220	Winter 2013	

1. Join *Students* with *Enrollment* (natural join)
2. Select only those tuples where *grade* = *'A'*

- Project the value of the *name* column for the resulting tuples

$\pi_{name}(\sigma_{grade='A'}(Students \bowtie Enrollment))$

name
Joe

Combining operations into queries

What are the names of students who got an A in any course?

Students

sid	name	gpa
1111	Joe	3.2
2222	Ann	4.0
3333	Mike	3.5

Enrollment

sid	did	cid	term	grade
1111	1	210	Fall 2012	A
2222	1	220	Winter 2013	

1. Select those tuples in *Enrollment* where *grade = 'A'*

- Join these tuples with *Students* (natural join)
- Project the value of the *name* column for the resulting tuples

$\pi_{name}(\textit{Students} \bowtie (\sigma_{grade='A'}\ \textit{Enrollment}))$

name
Joe

Examples

Sailors (sid, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18

Boats (bid, name, color)

bid	name	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Reserves (sid, bid, day)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List names of boats.

$$\pi_{name}(\text{Boats})$$

List ratings and ages sailors.

$$\pi_{rating,\ age}(\text{Sailors})$$

List names of sailors who are over 21 years old.

$$\pi_{name}(\sigma_{age>21}(\text{Sailors}))$$

List names of red boats.

$$\pi_{name}(\sigma_{color=red}(\text{Boats}))$$

Examples

Sailors (<u>sid</u>, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18

Boats (<u>bid</u>, name, color)

bid	name	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Reserves (<u>sid</u>, <u>bid</u>, <u>day</u>)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List names of sailors who have reserved Clipper.

$$\pi_{Sailors.name}\Big(Sailors \bowtie_{Sailors.sid=Reserves.sid} \big((\sigma_{Boat.name=Clipper}(Boats)) \bowtie_{Boats.bid=Reserves.bid} Reserves\big)\Big)$$

List names of boats that were reserved by Horatio.

$$\pi_{Boats.name}\Big((\sigma_{Sailor.name=Horatio}(Sailors)) \bowtie_{Sailors.sid=Reserves.sid} (Boats \bowtie_{Boats.bid=Reserves.bid} Reserves)\Big)$$

Renaming

Sometimes it is necessary to rename a relation, or columns in the relation. For this, we use the renaming operator.

$$\mathit{Boats}\ (\underline{\mathit{bid}}, \mathit{name}, \mathit{color}) \Rightarrow \mathit{Vessels}\ (\underline{\mathit{vid}}, \mathit{handle}, \mathit{shade})$$

$$\rho_{\mathit{Vessels(vid,\ handle,\ shade)}}\ (\mathit{Boats})$$

$$\mathit{Boats}\ (\underline{\mathit{bid}}, \mathit{name}, \mathit{color}) \Rightarrow \mathit{Barges}\ (\underline{\mathit{bid}}, \mathit{name}, \mathit{color})$$

$$\rho_{\mathit{Barges}}\ (\mathit{Boats})$$

A self-join

A self-join is a join that joins together tuples from two copies of the same table

List all possible heterosexual couples (girl name, boy name), where the boy is older than the girl.

People

id	name	age	gender
1	Ann	18	F
2	Jane	22	F
3	Mike	21	M
4	Dave	27	M

$$\pi_{Girls.name,\ Boys.name}\Big(\rho_{Girls}\big(\sigma_{gender=F}(People)\big) \bowtie_{Girls.age<Boys.age} \rho_{Boys}\big(\sigma_{gender=M}(People)\big)\Big)$$

Part 3: Summary

- Relational model basics
  - relation schema vs. instance
  - tuples (= rows), attributes (columns)
- Constraints in the relational model
  - domain
  - key
  - foreign key
  - not null / check
- Creating relations
- Analyzing data: relational algebra