SQL

Integrity Constraints in Create pk and ck, foreign keys

```
CREATE TABLE Student
   (sid CHAR(20),
    name CHAR(20)
    PRIMARY KEY (sid))
```

domain types: char(n), varchar(n), int, small int
numeric(p, d), real, double precision, float(n)

date: '2021-07-27' *4-digit year, month, day*
time: '09:00:30` *hours, minutes, seconds*
timestamp: '2001-7-27 09:00:30.75'
date plus time of day
interval: '1' day period of time

create domain person - name char(20) not null
domains are data types with optional constraints.

```
SELECT A1, A2, ... An        (π)
FROM r1, r2 ... rm           ×
WHERE P                      (σ)
```

by default, duplicates are not eliminated.

STRING MATCHING

LIKE is used for string matching
- _ stands for any one character
- % is 0 or more arbitrary characters

ORDERING

ORDER BY Name des/asc (asc default)
OR ORDER BY Year, Name
(first order by year, then name within year
OR ORDER BY c DESC, b ASC

SET OPERATIONS

Set Operations: union (∪), intersection (∩), except (−)

Each automatically eliminates duplicates.
To keep them, use UNION ALL
INTERSECT ALL
EXCEPT ALL
suppose a tuple occurs m times in r and
n times in s, then it occurs
- m + n times in r union all s
- min(m, n) times in r intersect all s
- max(0, m-n) times in r except all s

SQL Domain Types:

char(n). Fixed length character string with length *n*.
varchar(n). Variable length character strings, with maximum length *n*.
int. Integer (machine-dependent).
smallint. Small integer (machine-dependent).
numeric(p,d). Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point.
real, double precision. Floating point and double-precision floating point numbers, with machine-dependent precision.
float(n). Floating point number, with user-specified precision of at least *n* digits.

Null values are allowed in all the domain types.
To prohibit null values declare attribute to be **not null**
create domain in SQL-92 and 99 creates user-defined domain types
create domain *person-name* **char**(20) **not null**

4 Relational Algebra

σ, π, ×, −, ∪, ρ, ∩, ⋈, /, ←

- $\pi_{C,B}(R)$, order of C, B matters
- For ∪ ∩ −, r & s must have the same # of attributes, and attributes must have same domains.
- r × s, if end up w/ attributes w/ same name, attributes are referred to by pos.
- ρ(X, E) returns expression E under name X
  can also rename part of an expression
  ↳ ρ((starID → ID), $\pi_{starID, Name}$(MovieStar))
  ↳ ρ((1 → ID), $\pi_{starID, Name}$(MovieStar))
  ↳ ρ(GenderlessStars(ID, Nom), $\pi_{starID, Name}$(MS))
- $R \bowtie_c S = \sigma_c(R \times S)$ Y(A) ← π_cR
- Equijoin: special case of $R \bowtie_c S = \sigma_c(R \times S)$ where c contains only equalities
- Natural join: Equijoin on all common attr.
- Division expressing "for all", "for every"
  A/B contains all x tuples (MovieStars) s.t. for every y tuple (movies) in B, there is an x,y tuple in A
- $\pi_{A,B}(R)$ projection removes duplicate tuples!
  ↳ In SQL equivalent must use DISTINCT
  ↳ cols of same name of associate tables appear only once! × removes duplicates in SQL!

Division Statement

(a) $\pi_{JobNum, DriverENum}(Job) \;/\; \pi_{DriverENum}(\sigma_{JobNum = 301}(Job))$

```
select JobNum from Job A
where not exists(
      (select DriverENum from Job
       where JobNum = 301)

      except

      (select DriverENum from Job
       where Job.JobNum = A.JobNum)
)
```

Union (r ∪ s), Intersection (r ∩ s), Set-Difference (r − s)
1. r, s must have the same number of attributes
2. Attribute domains must be "compatible" (ex. 2nd column of r and s have the same domain of values)

MovieStar ∪ Singer

StarID	Name	Gender
1	Harrison Ford	Male
2	Vivian Leigh	Female
3	Judy Garland	Female
4	Sam Smith	Non-binary

MovieStar ∩ Singer

StarID	Name	Gender
3	Judy Garland	Female

Cross-product (r x s): Allows us to combine two relations.

Note: When r and s have attributes with the same name, refer to them by column position

Rename (ρ (X, E)): Assigns another name to a relation by returning expression E under the name X.

Note: Can also refer to attributes by position ρ ((StarID→ID), $\pi_{StarID, Name}$(MovieStar)) is the same as ρ ((1→ID), $\pi_{StarID, Name}$(MovieStar))

ρ ((1 → StarID1, 5 → StarID2), MovieStar x StarsIn)

Relational Algebra Examples

Find name of actors in Jaws and Spongebob "Joining with Relation"

Jaws ← $\pi_{name}((\sigma_{title = "Jaws"}(Movie)) \bowtie StarsIn) \bowtie MovieStar$

Spongebob ← $\pi_{name}((\sigma_{title = "Spongebob"}(Movie)) \bowtie StarsIn) \bowtie Moviestar$

Res ← Jaws ∩ Spongebob

Find name of actors in all movies "Has to be connected in some way"

$\pi_{name}(\pi_{starID, movieID}(StarsIn) / \pi_{movieID}(Movie)) \bowtie Moviestar)$

"all"

Find all profs in exactly all the commitees prof piper is in

R2 ← $\pi_{commname}(\sigma_{profname = 'piper'}(commitee))$ commitee piper is in

R3 ← $\pi_{commname}(commitee)$ − R2 commitee piper is not in

commitee / R2 − $\pi_{profname}$(commitee ⋈ R3)

profs touching piper's comitee — determine which profs are on commitees from R3

Find all profs who have offices in at least all the buildings that Piper has offices in

$\pi_{profname, building}(professor \bowtie department) / \pi_{building}(\sigma_{profname = 'piper'}(professor) \bowtie department)$

"all"

For each person, find all pizzas the person eats that aren't served by any pizzeria the Person frequents

Eats − $\pi_{name, pizza}$(Frequents ⋈ Serves)

Eats(name, pizza) — Pizzas the person frequents

Frequents(name, pizzeria)

Serves(pizza, pizzeria, price)

*If for each, has to do with domain — same domain

Find names of people who frequent only pizzarias serving at least one pizza they eat.

all the pizzarias someone frequents

$\pi_{name}(Person) - \pi_{name}(Frequents - \pi_{name, pizzaria}(Eats \bowtie Serves))$

without — all the pizzarias serving pizza someone eats

Find names of people who frequent all the pizzarias serving at least one pizza they eat.

$\pi_{name}(Person) - \pi_{name}(\pi_{name, pizzaria}(Eats \bowtie Serves) - Frequents)$

The most expensive pass cost

$\pi_{cost}(Pass) - \pi_{cost}(P_{cost \to newcost}(Pass) \bowtie_{newcost < cost} (Pass))$

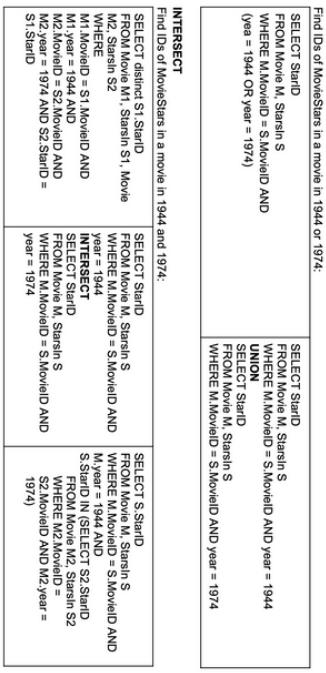
The min price book

$\pi_{isbn}(Book) - \pi_{b1.isbn}(\sigma_{b1.price > b2.price}(\rho(b1, Book) \times \rho(b2, Book)))$

Book w/ smallest price — books with a greater price then at least one other book

Find Pizzaria serving cheapest pepperoni pizza. For ties, return all of cheapest pepperoni pizzas.

$\pi_{pizzaria}(\sigma_{pizza = pepperoni}(Serves)) - \pi_{s1.pizzaria}(\sigma_{Note}(\rho(serves, s1) \times \rho(serves, s2))$



SQL EXAMPLES

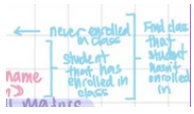
Find students who've taken all classes (division example)

1) With EXCEPT & NOT EXISTS

```
SELECT sname
FROM student s
WHERE NOT EXISTS ((SELECT c.name          -- all course names
                   FROM class c)
                  EXCEPT                  -- courses student is not enrolled in
                  (SELECT e.cname         -- courses student is enrolled in
                   FROM enrolled e
                   WHERE e.snum = s.snum))
```

2) Without EXCEPT using NOT EXISTS

```
SELECT sname
FROM student s
WHERE NOT EXISTS(SELECT c.name
                 FROM class c
                 WHERE NOT EXISTS (SELECT e.snum
                                   FROM enrolled e
                                   WHERE c.name = e.cname
                                   and e.snum = s.snum))
```



Find those majors for which their average age is the minimum over all majors

```
CREATE VIEW Temp(major, average) AS
   SELECT S.major, AVG(s.age) AS average
   FROM Student S
   GROUP BY major
```
grouping by major, getting average age

```
SELECT major, average
FROM Temp
WHERE average = (SELECT min(average)
                 FROM Temp)
```
selecting minimum over all majors

Get all students using natural join

```
SELECT *
FROM student s, enrolled e
WHERE s.snum = e.snum
```

```
SELECT *
FROM Student S NATURAL LEFT OUTER JOIN
Enrolled E
```

Find the names of all movie stars who've been in a movie

```
SELECT DISTINCT Name
FROM StarsIn s, Moviestar m
WHERE s.starsid = m.starid
```
Need distinct or else can get movie star name duplicates!

Find the departments that have more then one faculty member

```
SELECT DISTINCT f1.deptid
FROM Faculty f1, Faculty f2
WHERE f1.fid <> f2.fid AND f1.deptid = f2.deptid
```
seeing if different faculties have same department.

Find IDs of MovieStars who've been in a movie in 1944 or 1974

```
SELECT StarId
FROM Movie M, StarsIn S
WHERE M.MovieID = S.MovieID AND Year = 1944
UNION ALL keeps duplicates
SELECT StarId
FROM Movie M, StarsIn S
WHERE M.MovieID = s.movieID AND year=1974
```
For 1944 **and** 1974 use **INTERSECT**

equivalent to using year=1944 OR year=1974

Find the name & Age of the oldest student(s)

```
SELECT sname, age
FROM Student s2
WHERE NOT EXISTS (SELECT *
                  FROM student s1
                  WHERE s1. age > s2.age)
```
Find an s2 where there is no s1 older than them

Can also be written with any/all instead of exist/not exist

```
SELECT sname, age
FROM Student s2
WHERE s2.age >= ANY (SELECT age
                     FROM student s1)
```

Or with aggregate operators

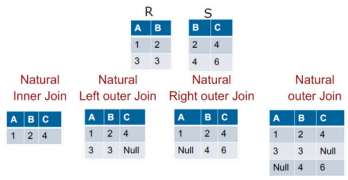
```
SELECT sname, age
FROM Student s2
WHERE s.age = (SELECT MAX(s2.age) FROM Student S2)
```

SET OPERATIONS PART 2

Set Operations used b/w 2 sets of tuples

- **UNION** usually used for **or**
- **INTERSECT** usually used for **and**
- **EXCEPT** usually used for **A but didn't B**
- **'have not been in'**

Natural Join
Default is INNER JOIN – only include matches.
FROM Student S NATURAL LEFT OUTER JOIN Enrolled E



NESTED QUERIES

NESTED QUERIES queries w/ another query embed

- SELECT, FROM, WHERE, HAVING, can itself contain a SQL query!
- using **IN** and **NOT IN**
  - ex. WHERE m.starid **IN** (select _ ...)
- Similar to nested loop evaluation
  - For each MovieStar tuple (FROM MovieStar), check qualification by computing subquery
- can replace **INTERSECT** queries using **IN**

EXISTS

EXISTS: true if set is not empty

- Used in combination w/ a subquery (t if at least 1 row)
- Can be used in SELECT, INSERT, UPDATE, DELETE
- Can also use NOT EXISTS

UNIQUE :True if there are no duplicates

op ANY, op ALL: ; operation can be >, <, =, <=, >=, <>

```
WHERE YEAR > ANY (SELECT year
                  FROM Movie
                  WHERE Title='Fargo')
```

Movies made after Fargo

Division in SQL

1. Better way w/ EXCEPT
2. Hard way without EXCEPT

AGGREGATE OPERATIONS

Functions for multiset of values of a col, return val
AVG, MIN, MAX, SUM, COUNT

This version eliminates duplicates before applying operation to A
COUNT(DISTINCT A), SUM(DISTINCT A), AVG(DISTINCT A)

```
SELECT country, city, count(*)
FROM customers
WHERE country LIKE '%a%'
GROUP BY country, city
HAVING count(*) < 3
ORDER BY count(*) DESC
```

this is for checking individual values

HAVING is for filtering sums/maxes/mins etc, is like a where condition for group by

VIEWS

Relations defined w/ a create table, statement existing in the physical layer

- Hide data from users, make queries easier, modularity

CREATE VIEW CourseWithFalls(dept, course #, mark) AS
SELECT c.dept, c.course#, mark
FROM Course C, Enrolled E
WHERE c.course# = e.course# and mark < 50

View updates must occur at base tables.

- DBMS restrict view updates only to some simple views on single tables (updatable views)

DROP VIEW <view name> does NOT affect any tuples in underlying relation

We can assign commands to DROP table for view.
DROP TABLE student RESTRICT/CASCADE

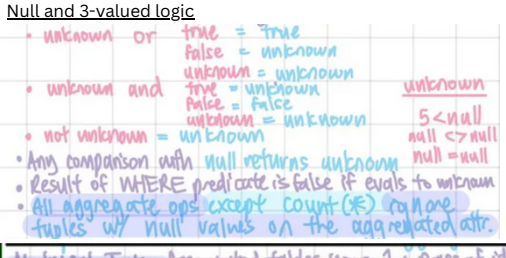
- drop table unless there's a view on it
- drop table & recursively drops any view referencing it

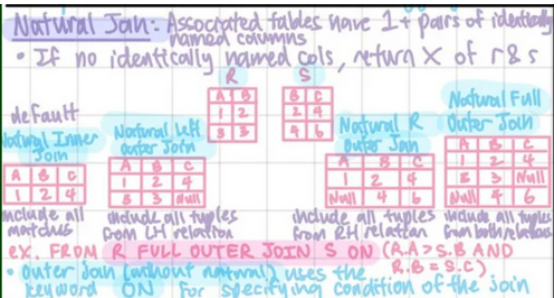
NULL VALUES

NULL Values: tuples can have null values. unknown or doesn't exist

- can use IS NULL (IS NOT NULL) in WHERE or others

Null and 3-valued logic





Enroll Student 51135593 into every class taught by faculty 90873519

INSERT, UPDATE, DELETE

Insert	Delete	Update
INSERT INTO Enrolled SELECT 51135593, name FROM Class WHERE fid = 90873519	DELETE FROM Student WHERE name = 'Smith	UPDATE Student SET age = age + 2 WHERE age < 98

Relational Algebra Part 2

Find all the professors who are in any of the committee's professor Piper is in

```
SELECT DISTINCT profname
FROM Committee
WHERE commname IN (SELECT *
                   FROM Committee
                   WHERE profname = 'Piper')
```

RA Statement

$\pi_{c2.profname}(\sigma_{c1.profname='Piper' \wedge c1.profname=c2.profname}(\rho_{c1}(committee) \times \rho_{c2}(committee)))$

Find all professors who are in at least all those commitees that professor Piper is in

```
SELECT DISTINCT c.profname
FROM Committee c
WHERE NOT EXISTS (SELECT commname
                  FROM Commitee a
                  WHERE profname = 'Piper')
                  EXCEPT
                  (SELECT commname
                  FROM committee a
                  WHERE a.profname=c.profname)
```

$committee / \pi_{commname}(\sigma_{profname='Piper'}(committee))$

Find all the enclosures that were never visited by the visitors born before 2000-01-01

```
SELECT E.id
FROM Employee E
EXCEPT
(SELECT EZ.id
 FROM Visitor v, Visits vs, Enclosure EZ
 WHERE v.visitorID = vs.passID AND vs.enclosureID = ez.id AND v.DateOfBirth < Date('2001-01-01')
```

Visitor(visitorID, firstName, lastName, dateofBirth)
Visits(passID, enclosureID)
Enclosure(id, latitude, longitude)

Find all the animals of species cat that are taken care of by 'Sam Smith'

```
SELECT employeeID
FROM TakesCareOf NATURAL INNER JOIN Employee
WHERE animalID IN (SELECT id from Animal WHERE Species=`Cat`) AND
                  firstName = 'Same' AND lastName = 'Smith'
```

Find the names of sailors who have reserved at least 2 boats

```
SELECT s.sname
FROM Sailors s, Reserves r1, Reserves r2
WHERE r1.sid=s.sid AND r2.sid=s.sid AND r1.bid <> r2.bid
```

Find the sailor id of the sailors with the highest rating

```
SELECT s.sid
FROM sailors s
WHERE s.rating >= (SELECT MAX(s2.rating) FROM Sailors s2)
```

Find the names of sailors who have reserved all boats whose name starts w/ "typhoon'

```
SELECT s.sname
FROM Sailors s
WHERE NOT EXISTS (SELECT b.bid FROM Boats b WHERE name LIKE "%typhoon%"
                  EXCEPT
                  (SELECT r.bid FROM Reserves r WHERE r.sid = s.sid))
```

"no boat w/ typhoon that we have not reserved"

Find the name & age of the oldest sailor

```
SELECT s.sname, s.age
FROM Sailor s
WHERE s.age = (SELECT MAX(s2.age) FROM Sailor s2)
```

Constraints for 1 table → use CHECK

```
CREATE TABLE Enrolled
  (snum  INTEGER,
   cname  CHAR(32),
   PRIMARY KEY (snum, cname),
   CONSTRAINT noR15
   CHECK ('R15' <>
          (SELECT c.room
           FROM   class c
           WHERE  c.name=cname)));
```

No one can be enrolled in a class, which is held in R15

Constraints over multiple tables → use ASSERTION

- Example: Write an assertion to enforce every student to be registered in at least one course.
  Student(snum, sname, major, standing, age)
  Enrolled(snum, cname)
  Class(name, meets_at, fid)

```
CREATE ASSERTION  Checkregistry
CHECK
( NOT EXISTS ((SELECT snum FROM student)
              EXCEPT
              (SELECT snum FROM enrolled)));
```

Nested Queries: A SELECT, FROM, WHERE, or HAVING clause can itself contain a SQL query

IN / NOT IN

Find IDs and names of female stars who have been in movie with ID 28:

```
SELECT  M.StarID, M.Name
FROM    MovieStar M
WHERE   M.Gender = 'female' AND
        M.StarID IN (SELECT  S.StarID
                     FROM    StarsIn S
                     WHERE   MovieID=28)
```

```
SELECT  S.StarID
FROM    StarsIn S
WHERE   MovieID=28
```

StarID
1026
1027

```
SELECT  M.StarID, M.Name
FROM    MovieStar M
WHERE   M.Gender = 'female' AND
        M.StarID IN
        (1026,1027)
```

EXISTS (true if subquery returns at least 1 row) / **NOT EXISTS**
ANY (true if condition is satisfied for at least 1 row in subquery) / **ALL**
Find the name and age of the oldest student(s):

```
SELECT sname, age
FROM Student s2
WHERE NOT EXISTS(SELECT *
                 FROM Student s1
                 WHERE s1.age >s2.age)
```

```
SELECT sname, age
FROM Student s2
WHERE s2.age >= ALL (SELECT age
                     FROM Student s1)
```

```
SELECT sname, age
FROM Student S
WHERE S.age=(SELECT MAX(S2.age
             FROM Student S2)
```

Unlike in RA, duplicates are not eliminated in SQL → specify using DISTINCT
SQL can rename attributes, ex. rename "StarID" to "ID" → SELECT StarID AS ID

String Matching:
- _ is for any one character
- % is for any zero or more characters
- Ex. WHERE cname LIKE '%System%'

Ordering
- ORDER BY AttributeName DESC / ASC (default)
- Can have nested ordering: ORDER BY Year, Name would first order by Year, then Name within years.

Joins (⋈)
R $\bowtie_{condition}$ **S** = $\sigma_{condition}$ (R×S)
Ex: **MovieStar** $\bowtie_{MovieStar.StarID < StarsIn.StarID}$ **StarsIn** will get all the tuples from the cross-product, MovieStar x StarsIn that satisfy the condition "MovieStar.StarID < StarsIn.StarID"
Equijoin: When the join condition contains only equalities: $\pi_{Name}((Movie \bowtie_{title = name \wedge StarID = MovieStar.StarID} MovieStar)$