CPSC 304
Introduction to Database Systems

Structured Query Language (SQL)

Textbook Reference
Database Management Systems: Chapter 5

Hassan Khosravi
Borrowing many slides from Rachel Pottinger

Databases: the continuing saga

When last we left databases…

- We had decided they were great things
- We knew how to conceptually model them in ER diagrams
- We knew how to logically model them in the relational model
- We knew how to normalize our database relations
- We could formally specify queries

Now: how do most people write queries? SQL!

Learning Goals

- Given the schemas of a relation, create SQL queries using: SELECT, FROM, WHERE, EXISTS, NOT EXISTS, UNIQUE, NOT UNIQUE, ANY, ALL, DISTINCT, GROUP BY and HAVING.
- Show that there are alternative ways of coding SQL queries to yield the same result. Determine whether or not two SQL queries are equivalent.
- Given a SQL query and table schemas and instances, compute the query result.
- Translate a query between SQL and RA.
- Comment on the relative expressive power of SQL and RA.
- Explain the purpose of NULL values and justify their use. Also describe the difficulties added by having nulls.
- Create and modify table schemas and views in SQL.
- Explain the role and advantages of embedding SQL in application programs.
- Write SQL for a small-to-medium sized programming application that requires database access.
- Identify the pros and cons of using general table constraints (e.g., CONSTRAINT, CHECK) and triggers in databases.

Coming up in SQL…

- Data Definition Language (reminder)
- Basic Structure
- Set Operations
- Aggregate Functions
- Null Values
- Nested Subqueries
- Modification of the Database
- Views
- Integrity Constraints
- Putting SQL to work in an application

The SQL Query Language

- Need for a standard since relational queries are used by many vendors
- Consists of several parts:
  - Data Definition Language (DDL)
    (a blast from the past (Chapter 3))
  - Data Manipulation Language (DML)
    - Data Query
    - Data Modification

Creating Tables in SQL(DDL) Revisited

- A SQL relation is defined using the **create table** command:

  create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

- *Integrity constraints can be:*
  - *primary and candidate keys*
  - *foreign keys*
- Example:

```
CREATE TABLE Student
     (sid     CHAR(20),
      name  CHAR(20),
      address  CHAR(20),
      phone  CHAR(8),
      major    CHAR(4),
      primary key (sid))
```

Domain Types in SQL Reference Sheet

- **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**

Date/Time Types in SQL Reference Sheet

- **date.** Dates, containing a (4 digit) year, month and date
  - E.g. **date** ‘2001-7-27’
- **time.** Time of day, in hours, minutes and seconds.
  - E.g. **time** ’09:00:30’ **time** ’09:00:30.75’
- **timestamp**: date plus time of day
  - E.g. **timestamp** ‘2001-7-27 09:00:30.75’
- **Interval**: period of time
  - E.g. Interval ‘1’ day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values
- Relational DBMS offer a variety of functions to
  - extract values of individual fields from date/time/timestamp
  - convert strings to dates and vice versa
  - For instance in Oracle (date is a timestamp):
    - TO_CHAR(date, format)
    - TO_DATE(string, format)
    - format looks like: ‘DD-Mon-YY HH:MI.SS’

Running Example (should look familiar)

Movie(<u>MovieID</u>, Title, Year)

StarsIn(<u>MovieID, StarID</u>, role)

MovieStar(<u>StarID</u>, Name, Gender)

Basic SQL Query

- SQL is based on set and relational operations
- A typical SQL query has the form:

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where P

  - A_is represent attributes
  - r_is represent relations
  - P is a predicate.

SELECT	*target-list*
FROM	*relation-list*
WHERE	*qualification*

π → SELECT clause
σ → WHERE clause
$\bowtie$ →FROM and WHERE clause

- The result of a SQL query is a table (relation)
- By default, duplicates are not eliminated in SQL relations, which are bags or multisets and not sets
- Let's compare to relational algebra…

Basic SQL/RA Comparison example 1

- Find the titles of movies

$$\pi_{Title}(Movie)$$

- In SQL, π is in the SELECT clause
- Select only a subset of the attributes

```
SELECT   Title
FROM     Movie
```

- Note duplication can happen!

Clicker Question: SQL projection

- Given the table scores: what is result of SELECT Score1, Score2 FROM Scores

Team1	Team2	Score1	Score2
Dragons	Tigers	5	3
Carp	Swallows	4	6
Bay Stars	Giants	2	1
Marines	Hawks	5	3
Ham Fighters	Buffaloes	1	6
Lions	Golden Eagles	8	12

- Which of the following rows is in the answer?

A. (1,2)
B. (5,3)
C. (8,6)
D. All are in the answer
E. None are in the answer

clickerprojection.sql

Clicker Question: SQL projection

- Given the table scores: what is result of SELECT Score1, Score2 FROM Scores

Team1	Team2	Score1	Score2
Dragons	Tigers	5	3
Carp	Swallows	4	6
Bay Stars	Giants	2	1
Marines	Hawks	5	3
Ham Fighters	Buffaloes	1	6
Lions	Golden Eagles	8	12

- Which of the following rows is in the answer?

A. (1,2)
B. (5,3) Correct
C. (8,6)
D. All are in the answer
E. None are in the answer

In SQL, σ is in *Where* clause

```
SELECT   *
FROM     Movie
WHERE    Year > 1939
```

You can use:

- attribute names of the relation(s) used in the FROM.
- comparison operators: =, <>, <, >, <=, >=
- apply arithmetic operations: rating*2
- operations on strings (e.g., "||" for concatenation).
- Lexicographic order on strings.
- Pattern matching: s LIKE p
- Special stuff for comparing dates and times.

Basic SQL/RA Comparison example 2

Find female movie stars

$$\sigma_{Gender = \text{'female'}} MovieStar$$

```
SELECT   *
FROM     MovieStar
WHERE    Gender = 'female'
```

Clicker Question: Selection

- Consider Scores(Team, Opponent, RunsFor, RunsAgainst) and query

  SELECT *
  FROM Scores
  WHERE
    RunsFor > 5

Team	Opponent	RunsFor	RunsAgainst
Dragons	Tigers	5	3
Carp	Swallows	4	6
Bay Stars	Giants	2	1
Marines	Hawks	5	3
Ham Fighters	Buffaloes	1	6
Lions	Golden Eagles	8	12
Tigers	Dragons	3	5
Swallows	Carp	6	4
Giants	Bay Stars	1	2
Hawks	Marines	3	5
Buffaloes	Ham Fighters	6	1
Golden Eagles	Lions	12	8

- Which tuple is in the result?

A. (Swallows, Carp, 6, 4)
B. (Swallows, Carp, 4)
C. (12)
D. (*)

clickerselection.sql

Clicker Question: Selection

- Consider Scores(Team, Opponent, RunsFor, RunsAgainst) and query
  SELECT *
  FROM Scores
  WHERE
  RunsFor > 5
- Which tuple is in the result?

A. (Swallows, Carp, 6, 4)
B. (Swallows, Carp, 4)
C. (12)
D. (*)

Team	Opponent	RunsFor	RunsAgainst
Dragons	Tigers	5	3
Carp	Swallows	4	6
Bay Stars	Giants	2	1
Marines	Hawks	5	3
Ham Fighters	Buffaloes	1	6
Lions	Golden Eagles	8	12
Tigers	Dragons	3	5
Swallows	Carp	6	4
Giants	Bay Stars	1	2
Hawks	Marines	3	5
Buffaloes	Ham Fighters	6	1
Golden Eagles	Lions	12	8

answer A

Selection & Projection – together forever in SQL

We can put these together:

- What are the names of female movie stars?

```
SELECT name
FROM MovieStar
WHERE Gender = 'female'
```

- What are the titles of movies from prior to 1939?
- ```
  SELECT title
  FROM Movie
  WHERE year < 1939
  ```
  ```

Selection example (dates)

events

name	date
A	1941-05-25
B	1942-11-15
C	1943-12-26
D	1944-10-25

Select *
From events
Where date < 19430000

name	date
A	1941-05-25
B	1942-11-15

Basic SQL/RA comparison example 3

- Find the person names and character names of those who have been in movies

- In order to do this we need to use joins. How can we do joins in SQL?
  - π → SELECT clause
    σ → WHERE clause
    $\bowtie$ → FROM and WHERE clause

Joins in SQL

```
SELECT   Role, Name
FROM     StarsIn s, MovieStar m
WHERE    s.StarID = m.StarID
```

- Cross product specified by From clause
- Can alias relations (e.g., “StarsIn s”)
- Conditions specified in where clause

Join Example

- Find the names of all movie stars who have been in a movie

```
SELECT Name
FROM StarsIn S, MovieStar M
WHERE S.StarID = M.StarID
```

Is this totally correct?

StarID	Name	Gender
1	Harrison Ford	Male
2	Vivian Leigh	Female
3	Judy Garland	Female

MovieID	StarID	Character
1	1	Han Solo
4	1	Indiana Jones
2	2	Scarlett O'Hara
3	3	Dorothy Gale

Harrison Ford will appear twice

Join Example

- Find the names of all movie stars who have been in a movie

```
SELECT Name
FROM StarsIn S, MovieStar M
WHERE S.StarID = M.StarID
```

Is this totally correct?

```
SELECT DISTINCT Name
FROM StarsIn S, MovieStar M
WHERE S.StarID = M.StarID
```

What if two movie stars had the same name?

- What if I run the following query?

```
SELECT DISTINCT StarID, Name
FROM StarsIn S, MovieStar M
WHERE S.StarID = M.StarID
```

Error: Column StarID is ambiguous

Clicker Question: Joins

- Consider R :

a	b
0	0
0	1
1	0
1	1

S:

a	b
0	0
0	1
1	0
1	1

T:

a	b
0	0
0	1
1	0
1	1

SELECT R.a, R.b, S.b, T.b
FROM R, S, T
WHERE R.b = S.a AND S.b <> T.b (note: <> == ‘not equals’)

Compute the results

Which of the following are true:

A. (0,1,1,0) appears twice.

B. (1,1,0,1) does not appear.

C. (1,1,1,0) appears once.

D. All are true

E. None are true

clickerjoin.sql

Clicker Question: Joins

- Consider R :

a	b
0	0
0	1
1	0
1	1

S:

a	b
0	0
0	1
1	0
1	1

T:

a	b
0	0
0	1
1	0
1	1

```
SELECT R.a, R.b, S.b, T.b
    FROM R, S, T
    WHERE R.b = S.a AND S.b <> T.b
```

(note: <> == 'not equals')

Compute the results

Which of the following are true:

A. (0,1,1,0) appears twice.

B. (1,1,0,1) does not appear.

C. (1,1,1,0) appears once.

D. All are true

E. None are true

True R(0,1) S(1,1), T(0,0)&
R(0,1), S(1,1), T(1,0),

False: R(1,1), S(1,0), T(0,1)

False: like A but use R(1, 1)

So how does a typical SQL query relate to relational algebra then?

SQL:

select A_1, A_2, ..., A_n
from r_1, r_2, ..., r_m
where P

Is approximately equal to

Relational algebra

$$\pi_{A1,\ A2,\ \ldots,\ An}(\sigma_P\ (r_1\ \ x\ r_2\ \ \ x\ \ \ldots\ \ x\ \ r_m))$$

Difference? Duplicates.

Remove them? Distinct

Using DISTINCT

- Find the names of actors who've been in at least one movie

```
SELECT   DISTINCT Name
FROM     StarsIn S, MovieStar M
WHERE    S.StarID = M.StarID
```

- Would removing DISTINCT from this query make a difference?

Distinction distinction

Why is it good; why is it bad?

- How many movies has Brad Pitt played?
  - You can’t do this query in RA with what you know
- Tricky to work with at times.

Clicker question: distinction

Consider the relation: Scores(Team, Opponent, RunsFor, RunsAgainst) and the query:

```
SELECT DISTINCT Team,
                RunsFor
FROM    Scores
```

Which is true:

A. 1 appears once
B. 5 appears twice
C. 6 appears 4 times
D. All are true
E. None are true

Team	Opponent	Runs For	Runs Against
Dragons	Tigers	5	3
Carp	Swallows	4	6
Bay Stars	Giants	2	1
Marines	Hawks	5	3
Ham Fighters	Buffaloes	1	6
Lions	Golden Eagles	8	12
Tigers	Dragons	3	5
Swallows	Carp	6	4
Giants	Bay Stars	1	2
Hawks	Marines	3	5
Buffaloes	Ham Fighters	6	1
Golden Eagles	Lions	12	8

Clicker question: distinction

Consider the relation: Scores(Team, Opponent, RunsFor, RunsAgainst) and the query:

SELECT DISTINCT Team,
RunsFor
FROM Scores

Which is true:

A. 1 appears once
B. 5 appears twice Correct
C. 6 appears 4 times
D. All are true
E. None are true

Team	Opponent	Runs For	Runs Against
Dragons	Tigers	5	3
Carp	Swallows	4	6
Bay Stars	Giants	2	1
Marines	Hawks	5	3
Ham Fighters	Buffaloes	1	6
Lions	Golden Eagles	8	12
Tigers	Dragons	3	5
Swallows	Carp	6	4
Giants	Bay Stars	1	2
Hawks	Marines	3	5
Buffaloes	Ham Fighters	6	1
Golden Eagles	Lions	12	8

clickerdistinction.sql

Renaming Attributes in Result

- SQL allows renaming relations and attributes using the **as** clause:

  old-name **as** *new-name*

- Example: Find the title of movies and all the characters in them, and rename "Role" to "Role1"

```
SELECT    Title, Role AS Role1
FROM      StarsIn S, Movie M
WHERE     M.MovieID = S.MovieID
```

Try select *; does not remove duplicate columns

Congratulations: You know select-project-join queries

- Very common subset to talk about
- Can do many (but not all) useful things

SQL is *declarative*, not procedural
how do we know? Lets see what
procedural would look like…

Conceptual Procedural Evaluation Strategy

1. Compute the cross-product of *relation-list*.
2. Discard resulting tuples if they fail *qualifications*.
3. Delete attributes that are not in *target-list*.
4. If DISTINCT is specified, eliminate duplicate rows.

Example of Conceptual Procedural Evaluation

SELECT Name

FROM MovieStar M, StarsIn S

WHERE S.StarID = M.StarID AND MovieID = 276

join   selection

MovieStar **X** StarsIn

(StarID)	Name	Gender	MovieID	(StarID)	Character
1273	Nathalie Portman	Female	272	1269	Leigh Anne Touhy
1273	Nathalie Portman	Female	273	1270	Mary
1273	Nathalie Portman	Female	274	1271	King George VI
1273	Nathalie Portman	Female	276	1273	Nina Sayers
…	…	…	…	…	…

New Students Example

- Class(<u>name</u>,meets_at,room,fid)
- Student(<u>snum</u>,sname,major,standing,age)
- Enrolled(<u>snum,cname</u>)
- Faculty(<u>fid</u>,fname,deptid)

Class Table

Name	Meets_at	Room	FID
Data Structures	MWF 10	R128	489456522
Database Systems	MWF 12:30-1:45	1320 DCL	142519864
Operating System Design	TuTh 12-1:20	20 AVW	489456522
Archaeology of the Incas	MWF 3-4:15	R128	248965255
Aviation Accident Investigation	TuTh 1-2:50	Q3	011564812
Air Quality Engineering	TuTh 10:30-11:45	R15	011564812
Introductory Latin	MWF 3-4:15	R12	248965255
American Political Parties	TuTh 2-3:15	20 AVW	619023588
Social Cognition	Tu 6:30-8:40	R15	159542516
Perception	MTuWTh 3	Q3	489221823
Multivariate Analysis	TuTh 2-3:15	R15	090873519
Patent Law	F 1-2:50	R128	090873519
Urban Economics	MWF 11	20 AVW	489221823
Organic Chemistry	TuTh 12:30-1:45	R12	489221823
Marketing Research	MW 10-11:15	1320 DCL	489221823
Seminar in American Art	M 4	R15	489221823
Orbital Mechanics	MWF 8 1320	DCL	011564812
Dairy Herd Management	TuTh 12:30-1:45	R128	356187925
Communication Networks	MW 9:30-10:45	20 AVW	141582651
Optical Electronics	TuTh 12:30-1:45	R15	254099823
Intoduction to Math	TuTh 8-9:30	R128	489221823

Student Table

SNUM	SNAME	MAJOR	ST	AGE
51135593	Maria White	English	SR	21
60839453	Charles Harris	Architecture	SR	22
99354543	Susan Martin	Law	JR	20
112348546	Joseph Thompson	Computer Science	SO	19
115987938	Christopher Garcia	Computer Science	JR	20
132977562	Angela Martinez	History	SR	20
269734834	Thomas Robinson	Psychology	SO	18
280158572	Margaret Clark	Animal Science	FR	18
301221823	Juan Rodriguez	Psychology	JR	20
318548912	Dorthy Lewis	Finance	FR	18
320874981	Daniel Lee	Electrical Engineering	FR	17
322654189	Lisa Walker	Computer Science	SO	17
348121549	Paul Hall	Computer Science	JR	18
351565322	Nancy Allen	Accounting	JR	19
451519864	Mark Young	Finance	FR	18
455798411	Luis Hernandez	Electrical Engineering	FR	17
462156489	Donald King	Mechanical Engineering	SO	19
550156548	George Wright	Education	SR	21
552455318	Ana Lopez	Computer Engineering	SR	19
556784565	Kenneth Hill	Civil Engineering	SR	21
567354612	Karen Scott	Computer Engineering	FR	18
573284895	Steven Green	Kinesiology	SO	19
574489456	Betty Adams	Economics	JR	20
578875478	Edward Baker	Veterinary Medicine	SR	21

Enrolled Table

SNUM	CNAME
112348546	Database Systems
115987938	Database Systems
348121549	Database Systems
322654189	Database Systems
552455318	Database Systems
455798411	Operating System Design
552455318	Operating System Design
567354612	Operating System Design
112348546	Operating System Design
115987938	Operating System Design
322654189	Operating System Design
567354612	Data Structures
552455318	Communication Networks
455798411	Optical Electronics
455798411	Organic Chemistry
301221823	Perception
301221823	Social Cognition
301221823	American Political Parties
556784565	Air Quality Engineering
99354543	Patent Law
574489456	Urban Economics

Faculty Table

FID	FNAME	DEPTID
142519864	I. Teach	20
242518965	James Smith	68
141582651	Mary Johnson	20
011564812	John Williams	68
254099823	Patricia Jones	68
356187925	Robert Brown	12
489456522	Linda Davis	20
287321212	Michael Miller	12
248965255	Barbara Wilson	12
159542516	William Moore	33
090873519	Elizabeth Taylor	11
486512566	David Anderson	20
619023588	Jennifer Thomas	11
489221823	Richard Jackson	33
548977562	Ulysses Teach	20

Running Examples

Movie(MovieID, Title, Year)
StarsIn(MovieID, StarID, role)
MovieStar(StarID, Name, Gender)

Student(snum,sname,major,standing,age)
Class(name,meets_at,room,fid)
Enrolled(snum,cname)
Faculty(fid,fname,deptid)

What kinds of queries can you answer so far?

Do we need DISTINCT?

- Find the names of all classes taught by Elizabeth Taylor

```
SELECT name
FROM Faculty f, class c
WHERE f.fid = c.fid and fname = 'Elizabeth Taylor'
```

Do we need f.fname?

- Find the student ids of those who have taken a course named “Database Systems”

```
SELECT snum
FROM enrolled e
WHERE cname = 'Database Systems'
```

What kinds of queries can you answer so far?

- Find the departments that have more than one faculty member (not equal <>)

```
SELECT DISTINCT f1.deptid
FROM faculty f1, faculty f2
WHERE f1.fid <>f2.fid AND
F1.deptid = f2.deptid
```

That is why renaming is important

f1

fid	fname	Deptid
90873519	Elizabeth Taylor	11
619023588	Jennifer Thomas	11
…	…	…

f2

fid	fname	Deptid
90873519	Elizabeth Taylor	11
619023588	Jennifer Thomas	11
…	…	…

A good example for using the same table twice in a query

Do I need Distinct?

What kinds of queries can you answer so far?

- Find the departments that have at least one faculty member

```
SELECT DISTINCT deptid
FROM faculty
```

String comparisons

- What are the student ids of those who have taken a course with “Database” in the name?

A string walks into a bar…

```
SELECT DISTINCT snum
FROM   enrolled
Where cname LIKE '%Database%'
```

- LIKE is used for string matching:
  - '_' stands for any one character and
  - '%' stands for 0 or more arbitrary characters.
- SQL supports string operations such as
  - concatenation (using "||")
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

Ordering of Tuples

- List in alphabetic order the names of actors who were in a movie in 1939

SELECT distinct Name
FROM Movie, StarsIn, MovieStar
WHERE Movie.MovieID = StarsIn.MovieID and StarsIn.StarID = MovieStar.StarID and year = 1939
ORDER BY Name

Order is specified by:

- **desc** for descending order
- **asc** for ascending order (default)
- E.g. **order by** *Name* **desc**

Clicker question: sorting

- Relation R has schema R(a,b,c). In the result of the query
  SELECT a, b, c
  FROM R
  ORDER BY c DESC, b ASC;
- What condition must a tuple *t* satisfy so that *t* **necessarily precedes** the tuple (5,5,5)? Identify one such tuple from the list below.

A. (3,6,3)
B. (1,5,5)
C. (5,5,6)
D. All of the above
E. None of the above

clickerorder.sql and clickerorder2.sql produce different ordering for 7,5,5 vs. 1,5,5

Clicker question: sorting

- Relation R has schema R(a,b,c). In the result of the query
  SELECT a, b, c
  FROM R
  ORDER BY c DESC, b ASC;
- What condition must a tuple *t* satisfy so that *t* **necessarily precedes** the tuple (5,5,5)? Identify one such tuple from the list below.

A. (3,6,3) — 3 < 5
B. (1,5,5) — Not specified
C. (5,5,6) — Right
D. All of the above
E. None of the above

Set Operations

- **union, intersect,** and **except** correspond to the relational algebra operations $\cup$, $\cap$, $-$.
- Each automatically eliminates duplicates;
  To retain all duplicates use the corresponding multiset versions:

  union all, intersect all and **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*

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

- **UNION:** Can union any two *union-compatible* sets of tuples (i.e., the result of SQL queries).

```
SELECT  StarID
FROM  Movie M, StarsIn S
WHERE  M.MovieID=S.MovieID AND
( year = 1944 OR year = 1974)
```

- The two queries though quite similar return different results, why?
  - Use UNION ALL to get the same answer

```
SELECT  StarID
FROM    Movie M, StarsIn S
WHERE   M.MovieID = S.MovieID AND
year = 1944
UNION
SELECT  StarID
FROM    Movie M, StarsIn S
WHERE   M.MovieID = S.MovieID AND
year = 1974
```

Set Operations: Intersect

- Example: Find IDs of stars who have been in a movie in 1944 *and* 1974.
- **INTERSECT:** Can be used to compute the intersection of any two *union-compatible* sets of tuples.
- In SQL/92, but some systems don't support it.

```
SELECT  StarID
FROM    Movie M, StarsIn S
WHERE   M.MovieID = S.MovieID AND
year = 1944
INTERSECT
SELECT  StarID
FROM    Movie M, StarsIn S
WHERE   M.MovieID = S.MovieID AND
year = 1974
```

Oracle does
MYSQL doesn't

Rewriting INTERSECT with Joins

- Example: Find IDs of stars who have been in a movie in 1944 *and* 1974 without using **INTERSECT.**

```
SELECT   distinct S1.StarID
FROM     Movie M1, StarsIn S1,
         Movie M2, StarsIn S2
WHERE
         M1.MovieID = S1.MovieID AND M1.year = 1944 AND
         M2.MovieID = S2.MovieID AND M2.year = 1974 AND
         S2.StarID = S1.StarID
```

Set Operations: EXCEPT

- Find the sids of all students who took Operating System Design but did not take Database Systems

```
Select snum
From enrolled e
Where cname = 'Operating System Design'
EXCEPT
Select snum
From enrolled e
Where cname = 'Database Systems'
```

Can we do it in a different way?
(We'll come back to this)

But what about…

- Select the IDs of all students who have not taken “Operating System Design”
  - One way to do is to find all students that taken “Operating System Design”.
  - Do all students MINUS those who have taken “Operating System Design”

Motivating Example for Nested Queries

- *Find ids and names of stars who have been in movie with ID 28:*

```
SELECT  M.StarID, name
FROM    MovieStar M, StarsIn S
WHERE   M.StarID = S.starID AND S.MovieID = 28;
```

- *Find ids and names of stars who have not been in movie with ID 28:*
  - *Would the following be correct?*

```
SELECT  M.StarID, name
FROM    MovieStar M, StarsIn S
WHERE   M.StarID = S.starID AND S.MovieID <> 28;
```

Nested Queries

- A very powerful feature of SQL:

```
Select A1,A2,…,An
From   R1,R2, …,Rm
Where  condition
```

- A nested query is a query that has another query embedded with it.
  - A **SELECT, FROM, WHERE, or HAVING** clause can itself contain an SQL query!
  - Being part of the WHERE clause is the most common

Nested Queries (IN/Not IN)

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

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

NOT IN

- To find stars who have *not* been in movie 28, use **NOT IN**.
- To understand nested query semantics, think of a *nested loops* evaluation:
  - *For each MovieStar tuple, check the qualification by computing the subquery.*

Nested Queries (IN/Not IN)

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

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

- In this example in inner query does not depend on the outer query so it could be computed just once.
- Think of this as a function that has no parameters.

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

StarID
1026
1027

```
SELECT  M.StarID, M.Name
FROM    MovieStar M
WHERE   M.StarID IN
(1026,1027)
```

Rewriting EXCEPT Queries Using In

- Using nested queries, find the sids of all students who took Operating System Design but did not take Database Systems

```
SELECT snum
FROM enrolled
WHERE cname = 'Operating System Design' and snum not in
          (SELECT snum
           FROM enrolled
           WHERE cname = 'Database Systems')
```

Rewriting INTERSECT Queries Using IN

Find IDs of stars who have been in movies in 1944 and 1974

```
SELECT S.StarID
FROM    Movie M, StarsIn S
WHERE  M.MovieID = S.MovieID AND M.year = 1944 AND
 S.StarID IN (SELECT  S2.StarID
                    FROM Movie M2, StarsIn S2
                    WHERE   M2.MovieID = S2.MovieID AND M2.year = 1974)
```

The subquery finds stars who have been in movies in 1974

We can also use alias M and S for the inner query and it would still work! (Locality)

Let's introduce one more schema

- We have high school students applying for college

```
College(cName,state,enrollment)
Student(sID,sName,GPA,sizeHS)
Apply(sID,cName,major,decision)
```

taken from Jennifer Widom's Stanford database course

Student table

```
Student(sID,sName,GPA,sizeHS)
```

insert into Student values (123, 'Amy', 3.9, 1000);
insert into Student values (234, 'Bob', 3.6, 1500);
insert into Student values (345, 'Craig', 3.5, 500);
insert into Student values (456, 'Doris', 3.9, 1000);
insert into Student values (567, 'Edward', 2.9, 2000);
insert into Student values (678, 'Fay', 3.8, 200);
insert into Student values (789, 'Gary', 3.4, 800);
insert into Student values (987, 'Helen', 3.7, 800);
insert into Student values (876, 'Irene', 3.9, 400);
insert into Student values (765, 'Jay', 2.9, 1500);
insert into Student values (654, 'Amy', 3.9, 1000);
insert into Student values (543, 'Craig', 3.4, 2000);

College Table

```
College(cName,state,enrollment)
```

insert into College values ('Stanford', 'CA', 15000);

insert into College values ('Berkeley', 'CA', 36000);

insert into College values ('MIT', 'MA', 10000);

insert into College values ('Cornell', 'NY', 21000);

Apply Table

Apply(<u>sID</u>,<u>cName</u>,<u>major</u>,decision)

```
insert into Apply values (123, 'Stanford', 'CS', 'Y');
insert into Apply values (123, 'Stanford', 'EE', 'N');
insert into Apply values (123, 'Berkeley', 'CS', 'Y');
insert into Apply values (123, 'Cornell', 'EE', 'Y');
insert into Apply values (234, 'Berkeley', 'biology', 'N');
insert into Apply values (345, 'MIT', 'bioengineering', 'Y');
insert into Apply values (345, 'Cornell', 'bioengineering', 'N');
insert into Apply values (345, 'Cornell', 'CS', 'Y');
insert into Apply values (345, 'Cornell', 'EE', 'N');
insert into Apply values (678, 'Stanford', 'history', 'Y');
insert into Apply values (987, 'Stanford', 'CS', 'Y');
insert into Apply values (987, 'Berkeley', 'CS', 'Y');
insert into Apply values (876, 'Stanford', 'CS', 'N');
insert into Apply values (876, 'MIT', 'biology', 'Y');
insert into Apply values (876, 'MIT', 'marine biology', 'N');
insert into Apply values (765, 'Stanford', 'history', 'Y');
insert into Apply values (765, 'Cornell', 'history', 'N');
insert into Apply values (765, 'Cornell', 'psychology', 'Y');
insert into Apply values (543, 'MIT', 'CS', 'N');
```

Our Three Running Examples

Movie(MovieID, Title, Year)
StarsIn(MovieID, StarID, role)
MovieStar(StarID, Name, Gender)

Student(snum,sname,major,standing,age)
Class(name,meets_at,room,fid)
Enrolled(snum,cname)
Faculty(fid,fname,deptid)

College(cName,state,enrollment)
Student(sID,sName,GPA,sizeHS)
Apply(sID,cName,major,decision)

Nested Queries Example

- Find IDs and names of students applying to CS (using both join and nested queries)

```
SELECT sID, sName
FROM Student
WHERE sID in (SELECT sID
              FROM Apply
                WHERE major = 'CS');
```

```
SELECT DISTINCT Student.sID, sName
FROM Student, Apply
WHERE Student.sID = Apply.sID and major = 'CS';
```

Do we need distinct?

Nested Query Example (tricky)

- Find names of students applying to CS (using both join and nested queries)

```
SELECT sName
FROM Student
WHERE sID in (SELECT sID
              FROM Apply
                 WHERE major = 'CS');
```

```
SELECT sName
FROM Student, Apply
WHERE Student.sID = Apply.sID and major = 'CS';
```

Do we need distinct?

Both with and without distinct is incorrect

Why are duplicates important?

- Find GPA of CS applicants (using both join and nested queries)

```
SELECT GPA
FROM Student
WHERE sID in (SELECT sID
              FROM Apply
              WHERE major = 'CS');
```

```
SELECT GPA
FROM Student, Apply
WHERE Student.sID = Apply.sID and major = 'CS';
```

Both with and without distinct is incorrect

SQL EXISTS Condition

- The SQL EXISTS condition is used in combination with a subquery and is considered to be met, if the subquery returns at least one row. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement.
- We can also use NOT EXISTS

Correlating Queries – Coming Up!

- Find the name of Colleges such that some other college is in the same state without nested queries.

```
SELECT C1.cName, C1.state
FROM College C1, College C2
WHERE C2.state = C1.state AND C2.cName <> C1.cName
```

Nested Queries with Correlation

In the examples seen so far, the inner subquery was always independent of the outer query

Find the name of Colleges such that some other college is in the same state

```
SELECT cName, state
FROM College C1
WHERE exists (SELECT *
                FROM College C2
                WHERE C2.state = C1.state AND
                      C2.cName <> C1.cName);
```

Think of this as passing parameters

- **EXISTS**: *returns true if the set is not empty*.
- Illustrates why, in general, subquery must be re-computed for each college tuple.
  - (For each college, check if there is another college in the same state

Exists Does work in MYSQL
Exists Does not work in oracle

SQL EXISTS Condition

- Using the EXISTS/ NOT EXISTS operations and correlated queries, 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)
```

More on Set-Comparison Operators

- We've already seen **IN and EXISTS**. Can also use **NOT IN**, **NOT EXISTS**.
- Also available: **op ANY**, **op ALL,** where op is one of: **>, <, =, <=, >=, <>**
- Find movies made after "Fargo"

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

Just returning one column

If we have multiple movies names Fargo then we can use ALL instead of ANY

Clicker nested question

Determine the result of:

```
SELECT Team, Day
FROM Scores S1
WHERE Runs <= ALL
    (SELECT Runs
    FROM Scores S2
    WHERE S1.Day = S2.Day )
```

Which of the following is in the result:

A. (Carp, Sun)
B. (Bay Stars, Sun)
C. (Swallows, Mon)
D. All of the above
E. None of the above

Scores:

Team	Day	Opponent	Runs
Dragons	Sun	Swallows	4
Tigers	Sun	Bay Stars	9
Carp	Sun	Giants	2
Swallows	Sun	Dragons	7
Bay Stars	Sun	Tigers	2
Giants	Sun	Carp	4
Dragons	Mon	Carp	6
Tigers	Mon	Bay Stars	5
Carp	Mon	Dragons	3
Swallows	Mon	Giants	0
Bay Stars	Mon	Tigers	7
Giants	Mon	Swallows	5

Clicker nested question

Clickernested.sql

Determine the result of:

```
SELECT Team, Day
FROM Scores S1
WHERE Runs <= ALL
    (SELECT Runs
    FROM Scores S2
    WHERE S1.Day = S2.Day )
```

Which of the following is in the result:

A. (Carp, Sun)
B. (Bay Stars, Sun)
C. (Swallows, Mon)
D. All of the above
E. None of the above

Correct

Scores:

Team	Day	Opponent	Runs
Dragons	Sun	Swallows	4
Tigers	Sun	Bay Stars	9
Carp	Sun	Giants	2
Swallows	Sun	Dragons	7
Bay Stars	Sun	Tigers	2
Giants	Sun	Carp	4
Dragons	Mon	Carp	6
Tigers	Mon	Bay Stars	5
Carp	Mon	Dragons	3
Swallows	Mon	Giants	0
Bay Stars	Mon	Tigers	7
Giants	Mon	Swallows	5

Team/Day pairs such that the team scored the minimum number of runs for that day.

Example

- Using the any or all operations, find the name and age of the oldest student(s)

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

```
SELECT sname, age
FROM student s2
WHERE not s2.age < any (SELECT age
                        FROM student s1)
```

You can rewrite queries that use any or all with queries that use exist or not exist

Clicker Question

- Consider the following SQL query

```
SELECT DISTINCT s1.sname, s1.age
FROM student s1, student s2
WHERE s1.age > s2.age
```

- This query returns
- A: The name and age of one of the oldest student(s)
- B: The name and age of all of the oldest student(s)
- C: The name and age of all of the youngest student(s)
- D: The name and age of all students that are older than the youngest student(s)
- E: None of the above

Clicker Question

- Consider the following SQL query

```
SELECT DISTINCT s1.sname, s1.age
FROM student s1, student s2
WHERE s1.age > s2.age
```

- This query returns
- A: The name and age of one of the oldest student(s)
- B: The name and age of all of the oldest student(s)
- C: The name and age of all of the youngest student(s)
- D: The name and age of all students that are older than the youngest student(s)
- E: None of the above

Division in SQL

Find students who've taken all classes.

(method 1)

```
SELECT sname
FROM   Student S
WHERE NOT EXISTS
        ((SELECT  C.name
           FROM  Class C)          All classes
           EXCEPT
           (SELECT  E.cname
Classes     FROM  Enrolled E
taken by S  WHERE  e.snum=S.snum))
```

The hard way (without EXCEPT: (method 2)

```
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))
```

select Student S such that ...
there is no Class C…

which is not taken by S

Method 2
Not tested on exams

Subqueries in From

```
Select A1,A2,…,An
From   R1,R2, …,Rm
Where  condition
```

- A subquery in the from clause returns a temporary table in database server's memory, which is used by the outer query for further processing.
  - A subquery in the FROM clause can't be correlated subquery as it can't be evaluated per row of the outer query.

Example

- Add scaled GPA based on sizeHS

```
SELECT sID, sName, GPA, sizeHS,
           GPA*(sizeHS/1000.0) as scaledGPA
FROM Student;
```

- Find students whose scaled GPA changes GPA by more than 1

```
SELECT sID, sName, GPA,  GPA*(sizeHS/1000.0) as scaledGPA
FROM Student
WHERE abs(GPA*(sizeHS/1000.0) - GPA) > 1.0;
```

```
SELECT *
FROM (SELECT sID, sName, GPA, GPA*(sizeHS/1000.0) as scaledGPA
        FROM Student) G
WHERE abs(scaledGPA - GPA) > 1.0;
```

GPA*(sizeHS/1000.0) is computed once

You're Now Leaving the World of Relational Algebra

- You now have many ways of asking relational algebra queries
  - For this class, you should be able write queries using all of the different concepts that we've discussed & know the terms used
  - In general, use whatever seems easiest, unless the question specifically asks you to use a specific method.
  - Sometimes the query optimizer may do poorly, and you'll need to try a different version, but we'll ignore that for this class.

Mind the gap

- But there's more you might want to know!
- E.g., "find the average age of students"
- There are extensions of Relational Algebra that cover these topics
  - We won't cover them
- We will cover them in SQL

Aggregate Operators

- These functions operate on the multiset of values of a column of a relation, and return a value

  AVG: average value
  MIN: minimum value
  MAX: maximum value
  SUM: sum of values
  COUNT: number of values

- The following versions eliminate duplicates before applying the operation to attribute A:

  COUNT (DISTINCT A)
  SUM (DISTINCT A)
  AVG (DISTINCT A)

```
SELECT count(distinct s.snum)
FROM enrolled e, Student S
WHERE e.snum = s.snum
```

```
SELECT count(s.snum)
FROM enrolled e, Student S
WHERE e.snum = s.snum
```

Aggregate Operators: Examples

students

```
SELECT  COUNT(*)
FROM    Student
```

Find name and age of the oldest student(s)

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

Can use table name S for both

Finding average age of SR students

```
SELECT  AVG (age)
FROM    Student
WHERE   standing='SR'
```

Aggregation examples

- Find the minimum student age

```
SELECT min(age)
FROM student;
```

- How many students have taken a class with “Database” in the title

```
SELECT count(distinct snum)
FROM enrolled
where cname like '%Database%'
```

Note: want distinct for when Students take 2 db classes

GROUP BY and HAVING

- Divide tuples into groups and apply aggregate operations to each group.
- Example: *Find the age of the youngest student for each major.*

For *i* = 'Computer Science',
'Civil Engineering'…

```
SELECT  MIN (age)
FROM    Student
WHERE   major = i
```

- Problem:

  We don't know how many majors exist, not to mention this is not good practice

Grouping Examples

Find the age of the youngest student who is at least 19, for each major

```
SELECT      major, MIN(age)
FROM        Student
WHERE       age >= 19
GROUP BY    major
```

Snum	Major	Age
115987938	Computer Science	20
112348546	Computer Science	19
280158572	Animal Science	18
351565322	Accounting	19
556784565	Civil Engineering	21
…	…	…

No Animal Science

Major	Age
Computer Science	19
Accounting	19
Civil Engineering	21
…	…

Grouping Examples with Having

Find the age of the youngest student who is at least 19, for each major with at least 2 such students

```
SELECT      major,  MIN(age)
FROM        Student
WHERE       age >= 19
GROUP BY    major
HAVING  COUNT(*) > 1
```

Snum	Major	Age
115987938	Computer Science	20
112348546	Computer Science	19
280158572	Animal Science	18
351565322	Accounting	19
556784565	Civil Engineering	21
…	…	…

Major	Age
Computer Science	19
Accounting	19
Civil Engineering	21
…	…

Major	
Computer Science	19

And there are rules

Find the age of the youngest student who is at least 19, for each major with at least 2 such students

```
SELECT     major, MIN(age)
FROM       Student
WHERE      age >= 19
GROUP BY   major
HAVING  COUNT(*) > 1
```

- Would it make sense if I select age instead of MIN(age)?
- *Would it make sense if I select snum to be returned?*
- *Would it make sense if I select major to be returned?*

Major	Age
Computer Science	19
Accounting	19
Civil Engineering	21
…	…

GROUP BY and HAVING (cont)

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
ORDER BY    target-list
```

- The *target-list* contains
  (i) attribute names
  (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
- Attributes in (i) must also be in *grouping-list*.
  - each answer tuple corresponds to a *group,*
  - *group* = a set of tuples with same value for all attributes in *grouping-list*
  - selected attributes must have a single value per group.
- Attributes in *group-qualification* are either in *grouping-list* or are arguments to an aggregate operator.

Conceptual Evaluation of a Query

1. compute the cross-product of *relation-list*
2. keep only tuples that satisfy *qualification* where
3. partition the remaining tuples into groups by the value of attributes in *grouping-list*
4. keep only the groups that satisfy *group-qualification* (expressions in *group-qualification* must have a *single value per group*!)
5. delete fields that are not in *target-list*
6. generate one answer tuple per qualifying group.

GROUP BY and HAVING (cont)

- Example1: *For each class, find the age of the youngest student who has enrolled in this class:*

```
SELECT     cname,  MIN(age)
FROM       Student S, Enrolled E
WHERE      S.snum= E.snum
GROUP BY   cname
```

- Example2: *For each course with more than 1 enrollment, find the age of the youngest student who has taken this class:*

```
SELECT     cname,  MIN(age)
FROM       Student S, Enrolled E
WHERE      S.snum = E.snum
GROUP BY   cname
HAVING     COUNT(*) > 1        ← per group qualification!
```

Clicker question: grouping

- Compute the result of the query:
  SELECT a1.x, a2.y, COUNT(*)
  FROM Arc a1, Arc a2
  WHERE a1.y = a2.x
  GROUP BY a1.x, a2.y
  (think of Arc as being a flight, and the query as asking for how many ways you can take each 2 hop plane trip)
  Which of the following is in the result?

x	y
1	2
1	2
2	3
3	4
3	4
4	1
4	1
4	1
4	2

A. (1,3,2)
B. (4,2,6)
C. (4,3,1)
D. All of the above
E. None of the above

clickergrouping.sql

Clicker question: grouping

- Compute the result of the query:

```
SELECT a1.x, a2.y, COUNT(*)
FROM   Arc a1, Arc a2
WHERE a1.y = a2.x
GROUP BY a1.x, a2.y
```

x	y
1	2
1	2
2	3
3	4
3	4
4	1
4	1
4	1
4	2

x	y	COUNT(*)
1	3	2
2	4	2
3	1	6
3	2	2
4	2	6
4	3	1

A. (1,3,2) (1,2)(2,3), (1,2)(2,3)

B. (4,2,6) 3 ways to do (4,1) and two ways to do (1,2)

C. (4,3,1) (4,2)(2,3)

D. All of the above Correct

E. None of the above

clickergrouping2.sql

Clicker question: grouping

- Compute the result of the query:
  SELECT a1.x, a2.y, COUNT(*)
  FROM Arc a1, Arc a2
  WHERE a1.y = a2.x
  GROUP BY a1.x, a2.y
  (The query asks for how many ways you can take each 2 hop plane trip. Which of the following is in the result?

A. (SFO,SEA,2)
B. (PIT,YVR,6)
C. (PIT,SEA,1)
D. All of the above correct
E. None of the above

FLIGHT:

origin	dest
SFO	YVR
SFO	YVR
YVR	SEA
SEA	PIT
SEA	PIT
PIT	SFO
PIT	SFO
PIT	SFO
PIT	YVR

Groupies of your very own

- Find the average age for each class standing (e.g., Freshman)

```
SELECT standing, avg(age)
FROM student
GROUP BY standing
```

- Find the deptID and # of faculty members for each department having an id > 20

(1)

```
SELECT count(*), deptid
FROM faculty
WHERE deptid > 20
GROUP BY deptid
```

(2)

```
SELECT count(*), deptid
FROM faculty
GROUP BY deptid
HAVING deptid > 20
```

Which one is correct?

A: just 1

B: just 2

C: both Correct

D: neither

Groupies of your very own

- Find the deptID and # of faculty members for each department with > 2 faculty (revisited!)

```
SELECT count(*), deptid
FROM faculty
GROUP BY deptid
HAVING count(*) > 2
```

Grouping Examples (cont')

For each standing, find the number of students who took a class with "System" in the title

```
SELECT  s.standing,  COUNT(DISTINCT s.snum) AS scount
FROM     Student S, enrolled E
WHERE   S.snum = E.snum and E.cname like '%System%'
GROUP BY  s.standing
```

- What if we do the following:
  (a) remove *E.cname like '%System%'* from the WHERE clause, and then
  (b) add a HAVING clause with the dropped condition?

```
SELECT  s.standing,  COUNT(DISTINCT s.snum) AS scount
FROM     Student S, enrolled E
WHERE   S.snum = E.snum
GROUP BY  s.standing
HAVING  E.cname like '%System%'
```

Not in groupby Error!

Clicker question: having

Suppose we have a relation with schema R(A, B, C, D, E). If we issue a query of the form:

```
SELECT ...
  FROM R
  WHERE ...
  GROUP BY B, E
  HAVING ???
```

What terms can appear in the HAVING condition (represented by ??? in the above query)? Identify, in the list below, the term that CANNOT appear.

A. A
B. B
C. Count(B)
D. All can appear
E. None can appear

Clicker question: having

Suppose we have a relation with schema R(A, B, C, D, E). If we issue a query of the form:

```
SELECT ...
  FROM R
  WHERE ...
  GROUP BY B, E
  HAVING ???
```

Any aggregated term can appear in HAVING clause. An attribute not in the GROUP-BY list cannot be unaggregated in the HAVING clause. Thus, B or E may appear unaggregated, and all five attributes can appear in an aggregation. However, A, C, or D cannot appear alone.

What terms can appear in the HAVING condition (represented by ??? in the above query)? Identify, in the list below, the term that CANNOT appear.

A. A

A cannot appear unaggregated

B. B

C. Count(B)

D. All can appear

E. None can appear

Grouping Examples (cont')

Find the age of the youngest student with age > 18, for each major with at least 2 students(of age > 18)

```
SELECT  S.major,  MIN(S.age)
FROM     Student S
WHERE   S.age > 18
GROUP BY  S.major
HAVING  count(*)  >1
```

Grouping Examples (cont')

Find the age of the youngest student with age > 18, for each major for which their average age is higher than the average age of all students across all majors.

```
SELECT  S.major,  MIN(S.age), avg(age)
FROM     Student S
WHERE   S.age > 18
GROUP BY  S.major
HAVING  avg(age)  >  (SELECT  avg(age)
                      FROM  Student)
```

Grouping Examples (cont')

Find the age of the youngest student with age > 18, for each major with at least 2 students(of any age)

```
SELECT  S.major,  MIN(S.age)
FROM      Student S
WHERE   S.age > 18
GROUP BY  S.major
HAVING  1  <  (SELECT  COUNT(*)
                      FROM  Student S2
                      WHERE  S.major=S2.major)
```

- Subqueries in the HAVING clause can be correlated with fields from the outer query.

Grouping Examples (cont')

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

```
SELECT major, avg(age)
FROM  student S
GROUP BY major
HAVING min(avg(age))
```

- WRONG, cannot use nested aggregation
  - One solution would be to use subquery in the From Clause

```
SELECT  Temp.major, Temp.average
FROM(SELECT  S.major, AVG(S.age) as average
          FROM  Student S
          GROUP BY S.major) AS Temp
WHERE Temp.average in (SELECT  MIN(Temp.average) FROM  Temp)
```

Hideously ugly
Not supported
in all systems

Grouping Examples (cont')

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

```
SELECT major, avg(age)
FROM  student S
GROUP BY major
HAVING min(avg(age))
```

- WRONG, cannot use nested aggregation
  - Another would be to use subquery with ALL in HAVING

```
SELECT major, avg(age)
FROM  student S
GROUP BY major
HAVING avg(age) <= all (SELECT  AVG(S.age)
                        FROM  Student S
                        GROUP BY S.major)
```

Easiest method would be to use Views

What are views

- Relations that are defined with a create table statement exist in the physical layer
  - do not change unless explicitly told so
- Virtual views do not physically exist, they are defined by expression over the tables.
    - Can be queries (most of the time) as if they were tables.

Why use views?

- Hide some data from users
- Make some queries easier
- Modularity of database
  - When not specified exactly based on tables.

Defining and using Views

- Create View <view name> As <view definition>
  - View definition is defined in SQL
  - From now on we can use the view almost as if it is just a normal table
- View V $(R_1,\ldots R_n)$
- query Q involving V
  - Conceptually
    - V $(R_1,\ldots R_n)$ is used to evaluate Q
  - In reality
    - The evaluation is performed over $R_1,\ldots R_n$

Defining and using Views

- Example: Suppose tables

  Course(Course#,title,dept)

  Enrolled(Course#,sid,mark)

```
CREATE VIEW CourseWithFails(dept, course#, mark) AS
    SELECT  C.dept, C.course#, mark
    FROM    Course C, Enrolled E
    WHERE   C.course# = E.course# AND mark<50
```

This view gives the dept, course#, and marks for those courses where someone failed

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given CourseWithFails, but not Course or Enrolled, we can find the course in which some students failed, but we can't find the students who failed.

Course(Course#,title,dept)
Enrolled(Course#,sid,mark)
VIEW CourseWithFails(dept, course#, mark)

View Updates

- View updates must occur at the base tables.
  - Ambiguous
  - Difficult

CourseWithFails(dept, course#, mark)

Course(<u>Course#</u>,title,dept)
Enrolled(<u>Course#</u>,<u>sid</u>,mark)

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

Example: UBC has one table for students. Should the CS Department be able to update CS students info? Yes, Biology students? NO
Create a view for CS to only be able to update CS students

View Deletes

- Drop View <view name>
  - Dropping a view does not affect any tuples of the in the underlying relation.

- How to handle DROP TABLE if there's a view on the table?
- DROP TABLE command has options to prevent a table from being dropped if views are defined on it:
  - DROP TABLE Student RESTRICT
    - drops the table, unless there is a view on it
  - DROP TABLE Student CASCADE
    - drops the table, and recursively drops any view referencing it

The Beauty of Views

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

With views:

```
Create View Temp(major, average) as
        SELECT     S.major, AVG(S.age) AS average
        FROM       Student S
        GROUP BY   S.major;

Select major, average
From Temp
WHERE average = (SELECT MIN(average) from Temp)
```

Without views:

```
  SELECT  Temp.major, Temp.average
FROM(SELECT  S.major, AVG(S.age) as average
     FROM  Student S
     GROUP BY S.major) AS Temp
WHERE Temp.average in (SELECT  MIN(Temp.average) FROM  Temp)
```

Hideously ugly

Clicker question: views

Suppose relation R(a,b,c):

Define the view *V* by:

```
CREATE VIEW V AS
SELECT a+b AS d, c
FROM R;
```

What is the result of the query:

```
SELECT d, SUM(c)
FROM V
GROUP BY d
HAVING COUNT(*) <> 1;
```

a	b	c
1	1	3
1	2	3
2	1	4
2	3	5
2	4	1
3	2	4
3	3	6

Identify, from the list below, a tuple in the result of the query:

A. (2,3)

B. (3,12)

C. (5,9)

D. All are correct

E. None are correct

Clickerview.sql

Clicker question: views

Suppose relation R(a,b,c):

Define the view *V* by:

```
CREATE VIEW V AS
SELECT a+b AS d, c
FROM R;
```

What is the result of the query:

```
SELECT d, SUM(c)
FROM V
GROUP BY d
HAVING COUNT(*) <> 1;
```

a	b	c
1	1	3
1	2	3
2	1	4
2	3	5
2	4	1
3	2	4
3	3	6

V

d	c
2	3
3	3
3	4
5	5
6	1
5	4
6	6

d	Sum(C)
3	7
5	9
6	7

Identify, from the list below, a tuple in the result of the query:

A. (2,3) Wrong. In view
B. (3,12)
C. (5,9) Right
D. All are correct
E. None are correct

Null Values

- Tuples may have a null value, denoted by *null*, for some of their attributes
- Value *null* signifies an unknown value or that a value does not exist.
- The predicate **IS NULL** (**IS NOT NULL**) can be used to check for null values.
  - E.g. *Find all student names whose age is not known.*

```
SELECT name
FROM Student
WHERE age IS NULL
```

- The result of any arithmetic expression involving *null* is *null*
  - E.g. 5 + *null* returns *null.*

Null Values and Three Valued Logic

- null requires a 3-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*, (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: *(true* **and** *unknown)* = *unknown, (false* **and** *unknown)* = *false, (unknown* **and** *unknown)* = *unknown*
  - NOT*:* *(***not** *unknown)* = *unknown*
  - "*P* **is unknown"** evaluates to true if predicate *P* evaluates to *unknown*
- Any comparison with *null* returns *unknown*
  - *E.g. 5 < null or null <> null or null = null*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes.

Round up

Round down

```
select count(*)
from class
```

```
select count(fid)
from class
```

Clicker null query

Determine the result of:

```
SELECT COUNT(*),
   COUNT(Runs)
FROM Scores
WHERE Team = 'Carp'
```

Which of the following is in the result:

A. (1,0)
B. (2,0)
C. (1,NULL)
D. All of the above
E. None of the above

Scores:

Team	Day	Opponent	Runs
Dragons	Sun	Swallows	4
Tigers	Sun	Bay Stars	9
Carp	Sun	NULL	NULL
Swallows	Sun	Dragons	7
Bay Stars	Sun	Tigers	2
Giants	Sun	NULL	NULL
Dragons	Mon	Carp	NULL
Tigers	Mon	NULL	NULL
Carp	Mon	Dragons	NULL
Swallows	Mon	Giants	0
Bay Stars	Mon	NULL	NULL
Giants	Mon	Swallows	5

Clicker null query

Start clickernull.sql

Determine the result of:

```
SELECT COUNT(*),
   COUNT(Runs)
FROM Scores
WHERE Team = 'Carp'
```

Which of the following is in the result:

A. (1,0)
B. (2,0) Right
C. (1,NULL)
D. All of the above
E. None of the above

Scores:

Team	Day	Opponent	Runs
Dragons	Sun	Swallows	4
Tigers	Sun	Bay Stars	9
Carp	Sun	NULL	NULL
Swallows	Sun	Dragons	7
Bay Stars	Sun	Tigers	2
Giants	Sun	NULL	NULL
Dragons	Mon	Carp	NULL
Tigers	Mon	NULL	NULL
Carp	Mon	Dragons	NULL
Swallows	Mon	Giants	0
Bay Stars	Mon	NULL	NULL
Giants	Mon	Swallows	5

Natural Join

- The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with same name of associate tables will appear once only.
- Natural Join : Guidelines
  - The associated tables have one or more pairs of identically named columns.
  - The columns must be the same data type.
  - Don't use ON clause in a natural join.

```
Select *
From student s natural join enrolled e
```

- Natural join of tables with no pairs of identically named columns will return the cross product of the two tables.

```
Select *
From student s natural join class c
```

More fun with joins

- What happens if I execute query:

  Select *

  From student s, enrolled e

  Where s.snum = e.snum
- To get *all* students, you need an *outer join*
- There are several special joins declared in the *from* clause:
  - Inner join – default: only include matches
  - Left outer join – include all tuples from left hand relation
  - Right outer join – include all tuples from right hand relation
  - Full outer join – include all tuples from both relations
- Orthogonal: can have natural join (as in relational algebra)

Example: SELECT *

FROM Student S NATURAL LEFT OUTER JOIN Enrolled E

More fun with joins examples

R

A	B
1	2
3	3

S

B	C
2	4
4	6

Natural Inner Join

A	B	C
1	2	4

Natural Left outer Join

A	B	C
1	2	4
3	3	Null

Natural Right outer Join

A	B	C
1	2	4
Null	4	6

Natural outer Join

A	B	C
1	2	4
3	3	Null
Null	4	6

Outer join (without the Natural) will use the key word on for specifying The condition of the join.

Outer join not implemented in MYSQL
Outer join is implemented in Oracle

Clicker outer join question

- Given:
  Compute:
  SELECT R.A, R.B, S.B, S.C, S.D
  FROM R FULL OUTER JOIN S
  ON (R.A > S.B AND R.B = S.C)

R(A,B) S(B,C,D)

A	B
1	2
3	4
5	6

B	C	D
2	4	6
4	6	8
4	7	9

- Which of the following tuples of R or S is dangling (and therefore needs to be padded in the outer join)?

A. (1,2) of R
B. (3,4) of R
C. (2,4,6) of S
D. All of the above
E. None of the above

Start clickerouter.sql

Clicker outer join question

- Given:
  Compute:
  SELECT R.A, R.B, S.B, S.C, S.D
  FROM R FULL OUTER JOIN S
  ON (R.A > S.B AND R.B = S.C)
- Which of the following tuples of R or S is dangling (and therefore needs to be padded in the outer join)?

A. (1,2) of R
B. (3,4) of R
C. (2,4,6) of S
D. All of the above
E. None of the above

A is correct

R(A,B) S(B,C,D)

A	B
1	2
3	4
5	6

B	C	D
2	4	6
4	6	8
4	7	9

A	B	B	C	D
3	4	2	4	6
5	6	4	6	8
1	2	NULL	NULL	NULL
NULL	NULL	4	7	9

Database Manipulation Insertion redux

- Can insert a single tuple using:
  INSERT INTO Student
  VALUES (53688, 'Smith', '222 W.15th ave', 333-4444, MATH)
- or

  INSERT INTO Student (sid, name, address, phone, major)
  VALUES (53688, 'Smith', '222 W.15th ave', 333-4444, MATH)

- Add a tuple to student with null address and phone:

  INSERT INTO Student (sid, name, address, phone, major)
  VALUES (33388, 'Chan', null, null, CPSC)

Database Manipulation Insertion redux (cont)

- Can add values selected from another table
- Enroll student 51135593 into every class taught by faculty 90873519

```
INSERT  INTO  Enrolled
SELECT  51135593, name
FROM Class
WHERE fid = 90873519
```

The select-from-where statement is fully evaluated before any of its results are inserted or deleted.

Database Manipulation
Deletion

- Note that only whole tuples are deleted.
- Can delete all tuples satisfying some condition (e.g., name = Smith):

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

Database Manipulation Updates

- Increase the age of all students by 2 (should not be more than 100)
- Need to write two updates:

```
UPDATE  Student
SET         age = 100
WHERE   age >= 98

UPDATE  Student
SET age = age + 2
WHERE age < 98
```

- Is the order important?

Integrity Constraints (Review)

- An IC describes conditions that every *legal instance* of a relation must satisfy.
  - Inserts/deletes/updates that violate IC's are disallowed.
  - Can ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- *Types of IC's*:
  - domain constraints,
  - primary key constraints,
  - foreign key constraints,
  - general constraints

General Constraints: Check

- We can specify constraints over a single table using table constraints, which have the form

Check conditional-expression

```
CREATE TABLE  Student
   ( snum  INTEGER,
     sname  CHAR(32),
     major  CHAR(32),
     standing CHAR(2)
     age  REAL,
     PRIMARY KEY  (snum),
     CHECK  ( age >= 10
              AND age < 100 );
```

Check constraints are checked when tuples are inserted or modified

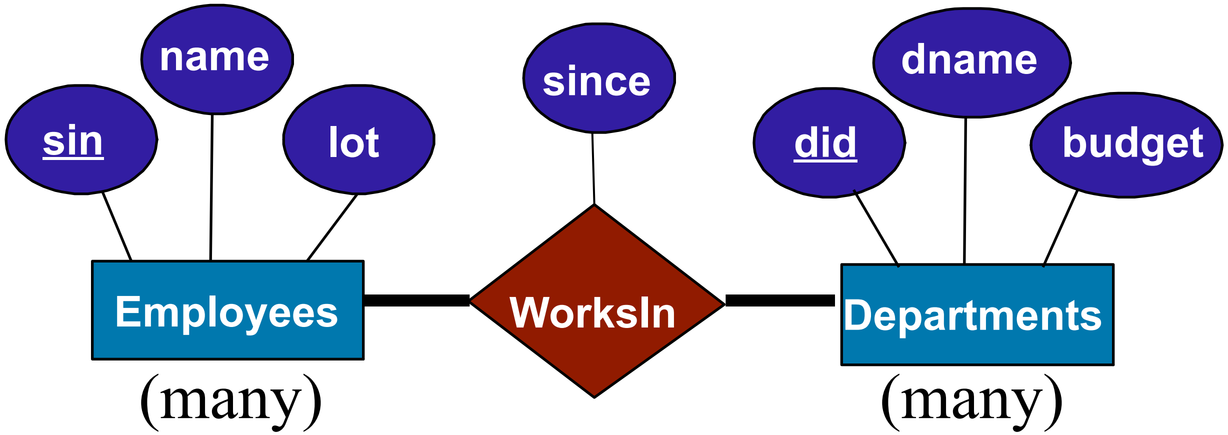
General Constraints: Check

- Constraints can be named
- Can use subqueries to express constraint
- Table constraints are associated with a single table, although the conditional expression in the check clause can refer to other tables

```
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 Relations: Remember this one?



- We couldn't express
  "every employee works in a department and every department has some employee in it"?
- Neither foreign-key nor not-null constraints in Works_In can do that.
- Assertions to the rescue!

Constraints Over Multiple Relations

- Cannot be defined in one table.
- Are defined as ASSERTIONs which are not associated with any table
- Example: *Every MovieStar needs to star in at least one Movie*

```
CREATE ASSERTION  totalEmployment
CHECK
( NOT EXISTS ((SELECT StarID FROM MovieStar)
                    EXCEPT
                   (StarID FROM StarsIn)));
```

Constraints Over Multiple Relations

- Example: Write an assertion to enforce every student to be registered in at least one course.

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

Triggers

- Trigger : a procedure that starts automatically if specified changes occur to the DBMS
- Active Database: a database with triggers
- A trigger has three parts:
  1. Event (activates the trigger)
  2. Condition (tests whether the trigger should run)
  3. Action (procedure executed when trigger runs)

Useful for project
Not tested on exams

- Database vendors did not wait for trigger standards! So trigger format depends on the DBMS
- NOTE: triggers may cause cascading effects.

Good way to shoot yourself in the foot

Triggers: Example (SQL:1999)

```
CREATE TRIGGER youngStudentUpdate
   AFTER INSERT ON Student
REFERENCING NEW TABLE NewStudent
FOR EACH STATEMENT
   INSERT INTO
      YoungStudent(snum, sname, major, standing, age)
      SELECT  snum, sname, major, standing, age
      FROM    NewStudent N
      WHERE   N.age <= 18;
```

event

newly inserted tuples

apply once per statement

action

Can be either before or after

That's nice. But how do we code with SQL?

- Direct SQL is rarely used: usually, SQL is embedded in some application code.
- We need some method to reference SQL statements.
- But: there is an *impedance mismatch* problem.
  - Structures in databases <> structures in programming languages
- Many things can be explained with the impedance mismatch.

The Impedance Mismatch Problem

The host language manipulates variables, values, pointers SQL manipulates relations.

There is no construct in the host language for manipulating relations. See https://en.wikipedia.org/wiki/Object-relational_impedance_mismatch

Why not use only one language?

- Forgetting SQL: “we can quickly dispense with this idea” [Ullman & Widom, pg. 363].
- SQL cannot do everything that the host language can do.

Database APIs

Rather than modify compiler, add library with database calls (API)

- Special standardized interface: procedures/ objects
- Passes SQL strings from language, presents result sets in a language-friendly way – solves that impedance mismatch
- Microsoft's *ODBC* is a C/C++ standard on Windows
- Sun's *JDBC* a Java equivalent
- API's are DBMS-neutral
  - a "driver" traps the calls and translates them into DBMS-specific code

A glimpse into your possible future: JDBC

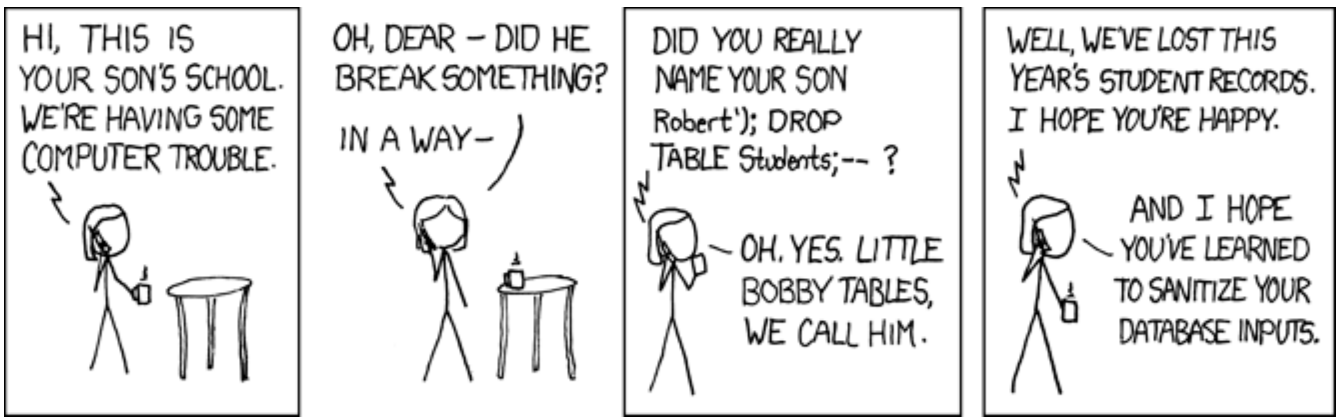
- JDBC supports a variety of features for querying and updating data, and for retrieving query results
- JDBC also supports metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes
- Model for communicating with the database:
  - Open a connection
  - Create a “statement” object
  - Execute queries using the Statement object to send queries and fetch results
  - Exception mechanism to handle errors

SQL API in Java (JDBC)

```
Connection con = // connect
   DriverManager.getConnection(url, ”login", ”pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT sname, age FROM Student";
ResultSet rs = stmt.executeQuery(query);
try { // handle exceptions
      // loop through result tuples
    while (rs.next()) {
        String s = rs.getString(“sname");
        Int n = rs.getFloat(“age");
        System.out.println(s + "   " + n);
    }
} catch(SQLException ex) {
    System.out.println(ex.getMessage ()
        + ex.getSQLState () + ex.getErrorCode ());
}
```

And now a brief digression

- Have you ever wondered why some websites don’t allow special characters?



Summary

- SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages.
- Relationally complete; in fact, significantly more expressive power than relational algebra.
- Consists of a data definition, data manipulation and query language.
- Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
  - In practice, users need to be aware of how queries are optimized and evaluated for best results.

Summary (Cont')

- NULL for unknown field values brings many complications
- SQL allows specification of rich integrity constraints (and triggers)
- Embedded SQL allows execution within a host language; cursor mechanism allows retrieval of one record at a time
- APIs such as ODBC and JDBC introduce a layer of abstraction between application and DBMS

Learning Goals Revisited

- Given the schemas of a relation, create SQL queries using: SELECT, FROM, WHERE, EXISTS, NOT EXISTS, UNIQUE, NOT UNIQUE, ANY, ALL, DISTINCT, GROUP BY and HAVING.
- Show that there are alternative ways of coding SQL queries to yield the same result. Determine whether or not two SQL queries are equivalent.
- Given a SQL query and table schemas and instances, compute the query result.
- Translate a query between SQL and RA.
- Comment on the relative expressive power of SQL and RA.
- Explain the purpose of NULL values and justify their use. Also describe the difficulties added by having nulls.
- Create and modify table schemas and views in SQL.
- Explain the role and advantages of embedding SQL in application programs.
- Write SQL for a small-to-medium sized programming application that requires database access.
- Identify the pros and cons of using general table constraints (e.g., CONSTRAINT, CHECK) and triggers in databases.