INFS7901
Database Principles

Structured Query Language (SQL)

Rocky Chen

SQL

- Standardize language, supported by all of the major commercial databases.

- Interactive use via graphical user interface or embedded in programs.

- Declarative, based on relational algebra

Following the Examples

MySQL: an open-source relational database management system

phpMyAdmin: a free software tool intended to handle the administration of MySQL over the Web (can download/install using XAMMP as in video in Blackboard/practical/project template…).

Databases used for providing examples: Available for download from the course website (examples.sql, run it on phpMyAdmin).

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

Borrowed from Rachel Pottinger from UBC

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

Borrowed from Jennifer Widom from Stanford

SQL Statements

- **Data Definition Language (DDL)**
  - Statements to define the database schema

Create
Alter
Drop

- **Data Manipulation Language (DML)**
  - Statements to manipulate the data

insert
delete
update
select

CREATE, ALTER and DROP TABLE statements

Basic SELECT Query

Set Operations

Aggregation, GROUP BY and HAVING

Data Definition Language (DDL)

- Data Definition Language (DDL) is one of the two main parts to the SQL language.

- DDL statements are used to define the database structure or schema.
  - CREATE - to create objects in the database
  - ALTER - alters the structure of the database
  - DROP - delete objects from the database

Creating Tables in SQL

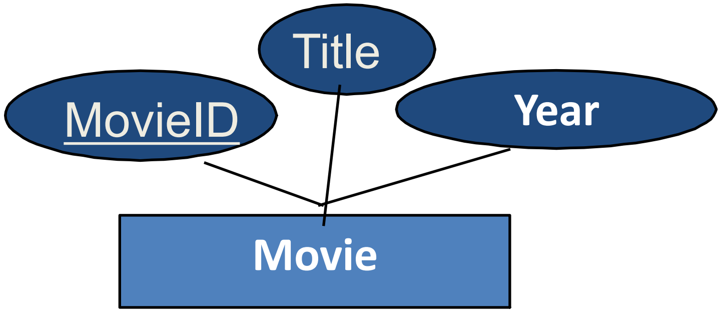
- **CREATE TABLE** statement creates a new relation, by specifying its name, attributes and constraints.
- The key, entity and referential integrity constraints are specified within the statement after the attributes have been declared.
- The domain constraint is specified for each attribute.
- Data type of an attribute can be specified directly or by declaring a domain (CREATE DOMAIN).

Creating Tables in SQL (DDL)

```
CREATE TABLE <table name>
    (<column name> <column type> [<attribute constraint>]
    {, <column name> <column type>  [<attribute constraint>] }
    [<table constraint> {, <table constraint>} ] )
```

- **<…>: mandatory**
- **[…]: optional**
- **{…}: support multiple inputs**

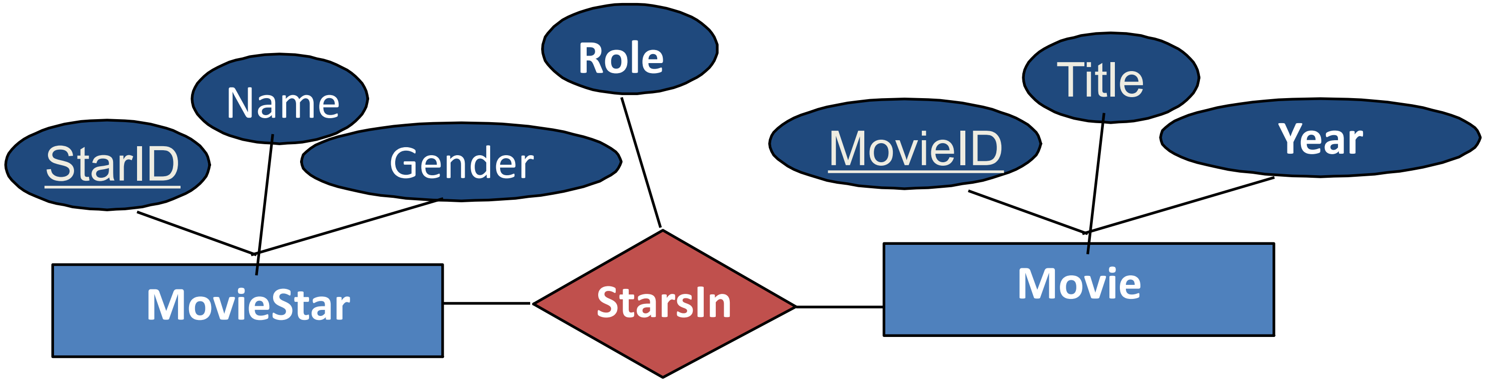
Creating an Entity Table



Movie [MovieID, Title, Year]

```
CREATE TABLE Movie
     (MovieID          INTEGER,
      Title            CHAR(20),
      Year             INTEGER,
      primary key (MovieID))
```

Creating a Relationship Table in SQL



StarsIn[MovieID, StarID, Role]

StarsIn.StarID → MovieStar.StarID

StarsIn.MovieID → Movies.MovieID

```
CREATE TABLE StarsIn (
  StarID      INTEGER,
  MovieID  INTEGER,
  Role         CHAR(20),
  PRIMARY KEY (StarID, MovieID),
  FOREIGN KEY (StarID) REFERENCES MovieStar(StarID),
  FOREIGN KEY (MovieID) REFERENCES Movies(MovieID))
```

Enforcing Referential Integrity

- MovieID in StarsIn is a foreign key that references Movies
  - StarsIn.MovieID → Movies.MovieID

- What should be done if a *movie tuple* is deleted?
  - Delete all roles that refer to it?
  - Disallow the deletion of the movie?
  - Set MID in StarsIn tuples that refer to it to null?
  - Set MID in StarsIn tuples that refer to it to default value?

Enforcing Referential Integrity

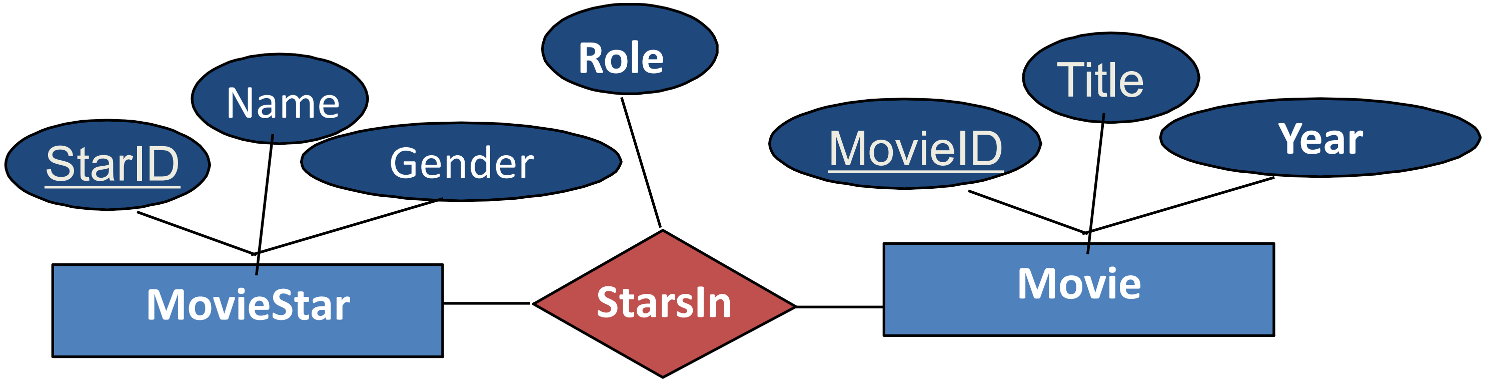
- A **referential triggered action** clause can be attached to a foreign key constraint, that specifies the action to take if a referenced tuple is deleted, or a referenced primary key value is modified.
- By default no action is taken and the delete/update is rejected.

- Other actions include the following:

ON DELETE SET NULL | SET DEFAULT | CASCADE

ON UPDATE SET NULL | SET DEFAULT | CASCADE

Creating Tables in SQL (DDL)



```
CREATE TABLE  StarsIn (
  StarID       INTEGER,
  MovieID  INTEGER,
  Role         CHAR(20),
  PRIMARY KEY (StarID, MovieID),
  FOREIGN KEY (StarID)  REFERENCES MovieStar (StarID)
       ON DELETE CASCADE
       ON UPDATE CASCADE,
  FOREIGN KEY (MovieID)  REFERENCES Movies (StarID)
        ON DELETE CASCADE
        ON UPDATE CASCADE )
```

Clicker Question

Consider the following table definition.

```
CREATE TABLE  Course_Selection  (sid  INTEGER, cid INTEGER,
               PRIMARY KEY (cid, sid),
               FOREIGN KEY (cid) REFERENCES Courses(cid)
                           ON DELETE CASCADE);
```

If sid = 1000 and cid = 5678 for a row in Table Course_Selection, choose the best answer

A. If the row for cid value 5678 in Courses is deleted, then the row (sid = 1000, cid = 5678) in Course_Selection is automatically deleted.

B. If row (sid = 1000, cid = 5678) in Course_Selection is deleted, then the row with cid=5678 in Courses is automatically deleted.

C. Both of the above.

Clicker Question

Consider the following table definition.

```
CREATE TABLE  Course_Selection  (sid  INTEGER, cid INTEGER,
                    PRIMARY KEY (cid, sid),
                    FOREIGN KEY (cid) REFERENCES Courses(cid)
                                   ON DELETE CASCADE);
```

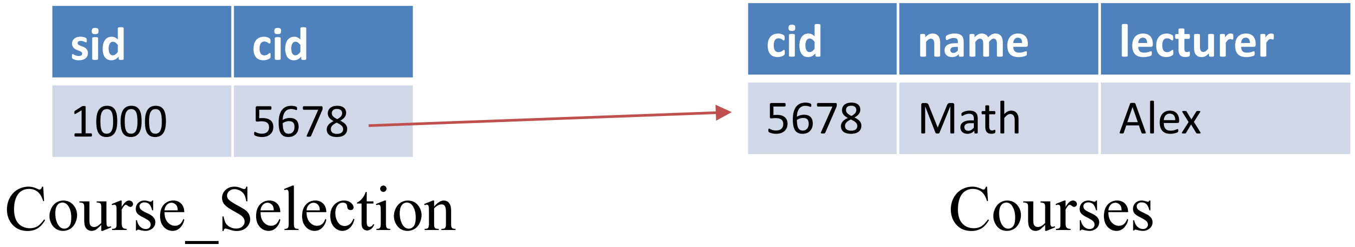
If sid = 1000 and cid = 5678 for a row in Table Course_Selection, choose the best answer

A. If the row for cid value 5678 in Courses is deleted, then the row (sid = 1000, cid = 5678) in Course_Selection is automatically deleted.

A is the correct answer

B. If row (sid = 1000, cid = 5678) in Course_Selection is deleted, then the row with cid=5678 in Courses is automatically deleted.

C. Both of the above.



sid	cid
1000	5678

Course_Selection

cid	name	lecturer
5678	Math	Alex

Courses

ALTER TABLE

- ALTER TABLE command is used for *schema evolution*, that is the definition of a table created using the CREATE TABLE command, can be changed using the ALTER TABLE command
- Alter table actions include
  - Adding or dropping a column.
  - Changing a column definition.
  - Adding or dropping constraints.

ALTER TABLE Syntax

ALTER TABLE <table name>
ADD <column name> <column type>
[<attribute constraint>] {, <column name>
<column type> [<attribute constraint>] }
| **DROP** <column name> [CASCADE]
| **ALTER** <column name> <column-options>
| **ADD** <constraint name> <constraint-options>
| **DROP** <constraint name> [CASCADE];

DROP TABLE

- DROP TABLE
  - Drops all constraints defined on the table including constraints in other tables which reference this table.
  - Deletes all tuples within the table.
  - Removes the table definition from the system catalog.

- DROP TABLE Syntax

```
DROP TABLE [IF EXISTS]
tbl_name [, tbl_name] ...
[RESTRICT | CASCADE]
```

CREATE, ALTER and DROP TABLE statements

Basic SELECT Query

Set Operations

Aggregation, GROUP BY and HAVING

Data Manipulation Language (DML)

- Data Manipulation Language (DML) is the other main part of the SQL language.

- DML statements are used for managing data within schema objects.
  - SELECT - retrieve data from a database.
  - INSERT - insert data into a table.
  - UPDATE - updates existing data within a table.
  - DELETE – deletes records from a table.

Basic SELECT Query

- In the SELECT statement, users specify what the result of the query should be, and the DBMS decides the operations and order of execution, thus SQL queries are "Declarative".

- The result of a SQL query is a table (relation).

- Note that the SQL SELECT statement has NO relationship to the SELECT operation of relational algebra !

Basic SELECT Query

- Selection (WHERE clause)
  - Horizontal scanner to select tuples from given collection of tuples.
- Projection (SELECT clause)
  - Vertically select the attributes of given collection of tuples.
- Join (FROM clause)
  - Combine tuples from different relations for the search purposes.
- Sorting (ORDER clause)
  - Order the resulting tuples according to the given sort key.

SELECT Basic Syntax

```
SELECT <attribute list>
FROM <table list>
[WHERE <condition>] ;
```

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

Projection in SQL

- Projection (SELECT clause)
  - Vertically select the attributes of given collection of tuples.

```
SELECT [DISTINCT] (attribute list | * )
FROM <table list>
[WHERE <condition>];
```

- **Distinct**: By default, duplicates are not eliminated in SQL relations. Use of distinct will eliminate duplicates and enforce set semantics.
- *: acts as a *wild card*, selecting all of the columns in the table.

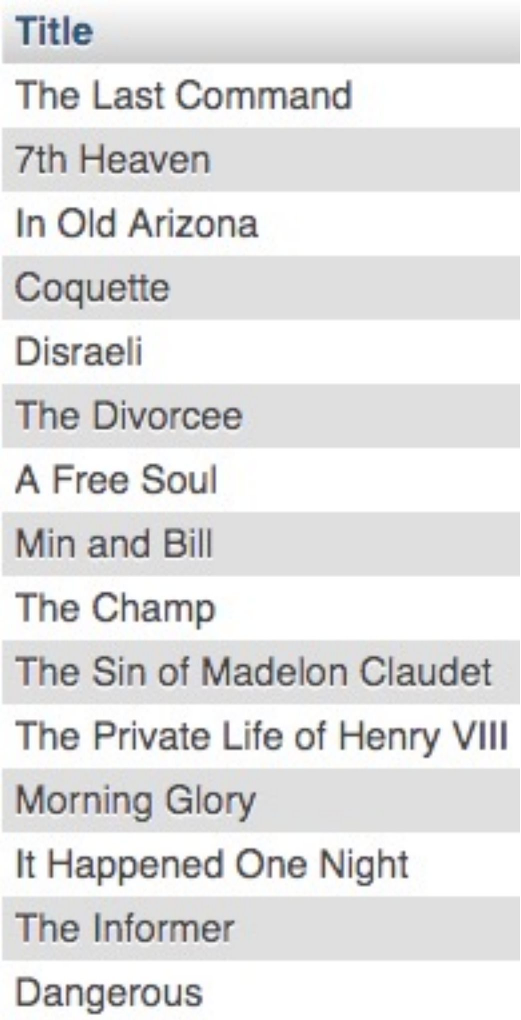
Projection Example

- Find the titles of movies.

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

Query

```
SELECT   Title
FROM      Movie
```



Title
The Last Command
7th Heaven
In Old Arizona
Coquette
Disraeli
The Divorcee
A Free Soul
Min and Bill
The Champ
The Sin of Madelon Claudet
The Private Life of Henry VIII
Morning Glory
It Happened One Night
The Informer
Dangerous

Clicker Question: SQL Projection

- Consider the given table and SQL query.

```
SELECT Score1, Score2
FROM Scores
```

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 one of the following tuples is in the result?

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

Clicker Question: SQL Projection

- Consider the given table and SQL query.

```
SELECT Score1, Score2
FROM Scores
```

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 one of the following tuples is in the result?

A. (1,2)

B. (5,3) Correct Answer

C. (8,6)

D. All are in the answer

E. None are in the answer

clickerprojection.sql

Projection and Duplicates

- Find all the years where a movie was produced.

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

Query

```
SELECT   Year
FROM       Movie
```

Query

```
SELECT DISTINCT Year
FROM       Movie
```

Removes duplicates

Clicker Question on Distinction

- Consider the given table and SQL query.

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

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

Which is true:

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

Clicker Question on Distinction

- Consider the given table and SQL query.

clickerdistinction.sql

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

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

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

Projection and Expressions

- SQL queries can also evaluate expressions and return the value of these expressions together with the projected attributes.
- Expressions use standard arithmetic operators (+, -, *, /) on numeric values or attributes with numeric domains.

Query

```
SELECT   Year
FROM      Movie
```

Query

```
SELECT   Year+2
FROM      Movie
```

Selection in SQL

- Selection (WHERE clause)
  - Horizontal scanner to select tuples from given collection of tuples.

SELECT <attribute list>
FROM <table list>
[WHERE join condition and **search_condition**]

<search condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

Select Search Example

- Find all of the male stars.

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

Query

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

Selection Example with Dates

- Find events that have occurred before 1943

events

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

Query

```
SELECT *
FROM events
WHERE date < 19430000
```

Results

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

Selection & Projection – Together Forever

- What are the names of the 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
```

Complex WHERE Conditions

- Find the title of all of the movies that contain “sin”

```
SELECT    *
FROM      Movie
Where Title like "%sin%"
```

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

Complex WHERE Conditions

- Substring Comparisons
  - LIKE
    - … WHERE Address LIKE ‘%St Lucia%’
    - … WHERE StrDate LIKE ‘_ _ / 0 5 / _ _’
  - IN
    - … WHERE LName IN (‘Jones’, ‘Wong’, ‘Harrison’)
  - IS
    - … WHERE DNo IS NULL
- Arithmetic Operators and Functions
  - +, -, *, /, date and time functions, etc.
    - … WHERE Salary * 2 > 50000
    - … WHERE Year(Sys_Date - Bdate) > 55
  - BETWEEN
    - … WHERE Salary BETWEEN 10000 AND 30000

Join in SQL

- Join (FROM clause)
  - Combine tuples from different relations for the search purposes.

```
SELECT <attribute list>
FROM <table list of more than one table>
[WHERE join condition and search_condition]
```

- <join condition> corresponds to a join condition in Relational Algebra.
- Alias for Table names are used to give a table a temporary name to make the query more readable.
  - e.g., FROM StarsIn S

Join in SQL

- Joining R1 and R2 on their shared attribute B:
  - each tuple of R1 is concatenated with every tuple in R2 having the same values on the join attributes.

```
SELECT A, R1.B, C
FROM R1, R2
WHERE R1.B = R2.B
```

R_1

A	B
1	2
4	5
7	2

R_2

B	C
2	3
5	6
2	8

$R_1 \bowtie R_2$ →

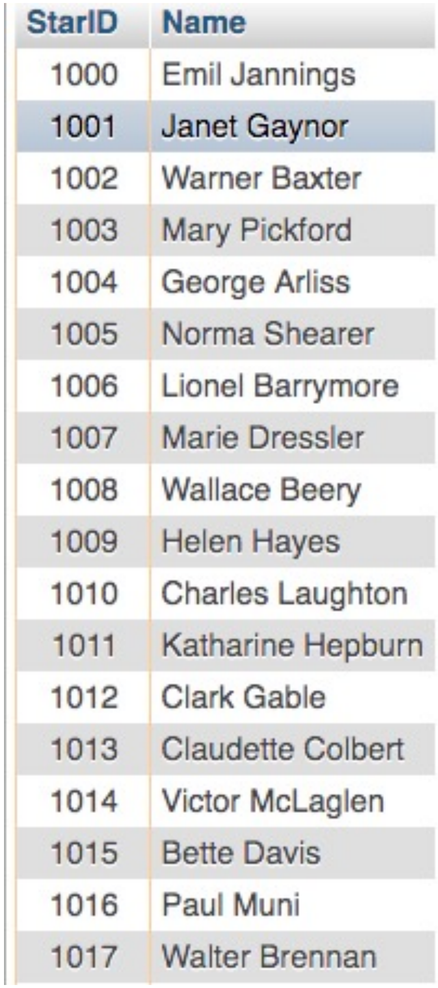
A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Join Example with Duplication

- Find the ids and names of all movie stars who have acted in a movie.

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

SELECT **DISTINCT** S.StarID, Name
FROM StarsIn S, MovieStar MS
WHERE S.StarID = MS.StarID



StarID	Name
1000	Emil Jannings
1001	Janet Gaynor
1002	Warner Baxter
1003	Mary Pickford
1004	George Arliss
1005	Norma Shearer
1006	Lionel Barrymore
1007	Marie Dressler
1008	Wallace Beery
1009	Helen Hayes
1010	Charles Laughton
1011	Katharine Hepburn
1012	Clark Gable
1013	Claudette Colbert
1014	Victor McLaglen
1015	Bette Davis
1016	Paul Muni
1017	Walter Brennan

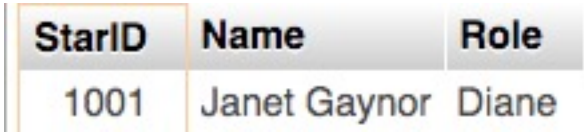
Join Example

- Find the ids, names and characters of all movie stars who have been in the movie with MovieID 1

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

Query

```
SELECT S.StarID, Name, Role
FROM StarsIn S, MovieStar MS
WHERE S.StarID = MS.StarID and
S.MovieID = 1
```



StarID	Name	Role
1001	Janet Gaynor	Diane

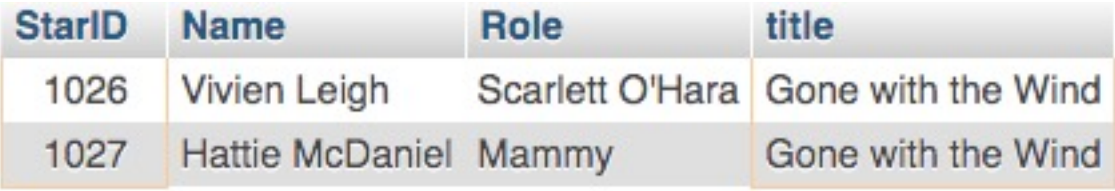
Join Example - Complex Conditions

- Find the ids, names and characters of all movie stars who have been in the movie titled ‘Gone with the Wind’.

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

Query

```
SELECT S.StarID, Name, Role, M.title
FROM StarsIn S, MovieStar MS, Movie M
WHERE S.StarID = MS.StarID and
 S.MovieID = M.MovieID and
M.title like "Gone with the Wind"
```



StarID	Name	Role	title
1026	Vivien Leigh	Scarlett O'Hara	Gone with the Wind
1027	Hattie McDaniel	Mammy	Gone with the Wind

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.

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

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

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

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

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

D. All are true

E. None are true

Renaming Attributes

- 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

Sorting in SQL

- Sorting (ORDER clause)
  - Order the resulting tuples according to the given sort key.

```
SELECT [DISTINCT] (attribute/expression list | * )
FROM <table list>
[WHERE [join condition and]  search_condition]
[ORDER BY column_name [ASC|DESC] {, column-name [ASC|DESC]}];
```

Order is specified by:

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

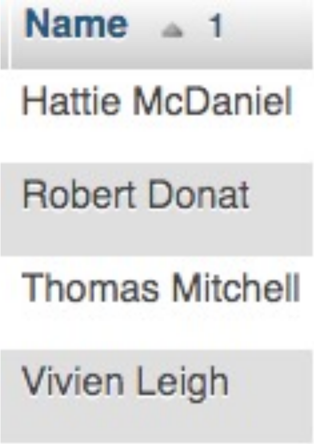
Ordering of Tuples

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

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

Query

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



Name ▲ 1
Hattie McDaniel
Robert Donat
Thomas Mitchell
Vivien Leigh

Clicker question: sorting

- Consider the following 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 (i.e., goes before)** 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

Clicker question: sorting

- Consider the following 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 (i.e., goes before)** 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) Correct

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

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.
5. If ORDER BY is specified, sort the results.

CREATE, ALTER and DROP TABLE statements

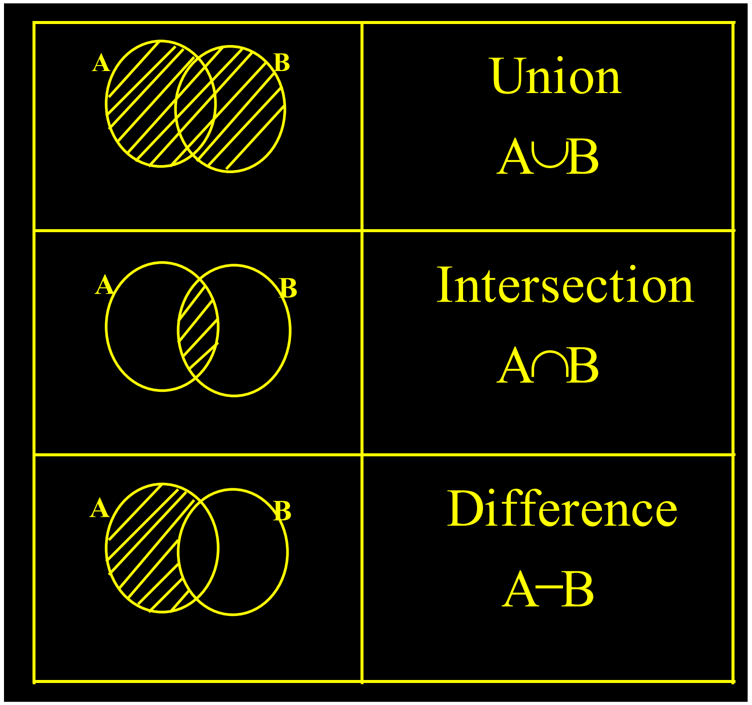
Basic SELECT Query

Set Operations

Aggregation, GROUP BY and HAVING

Basic Set Operators

- Relation is a ***set*** of tuples (no duplicates).
- Set theory, and hence elementary set operators also apply to relations
  - – UNION
  - – INTERSECTION
  - – DIFFERENCE



	Union A∪B
	Intersection A∩B
	Difference A−B

Set Operations

- **Union, intersect,** and **except** correspond to the relational algebra operations ∪, ∩, −.
- 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*

Union Compatibility

Two relations *R1(A1, A2, ..., An)* and *R2(B1, B2, ..., Bn)* are *union compatible* iff:

- They have the same degree n, (number of columns).
- Their columns have corresponding domains, i.e dom(Ai) = dom(Bi) for 1 ≤ i ≤ n

- Note that although domains need to correspond they do not have to have the same name.

Set Operations: Union

- **UNION:** Produces a relation that includes all tuples that appear only in R1, or only in R2, or in both R1 and R2.
  - Duplicate Tuples are eliminated if UNION ALL is not used.
  - R1 and R2 must be union compatible.

```
SELECT ...
UNION [ALL] SELECT ...
[UNION [ALL] SELECT ...]
```

Union Example

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

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

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

```
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
```

- Are the queries the same?

Intersection in SQL

- Intersection: Produces a relation that includes the tuples that appear in both R1 and R2.
  - Duplicate Tuples are eliminated if INTERSECT ALL is not used.
  - R1 and R2 must be union compatible.

```
SELECT ...
INTERSECT [ALL] SELECT ...
[INTERSECT [ALL] SELECT ...]
```

Intersect Example

- Find IDs of stars who have been in a movie in 1944 *and* 1974.

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

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

```
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
```

INTERSECT is part of the SQL standard, but is not implemented in MySQL.

Rewriting INTERSECT with Joins

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

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

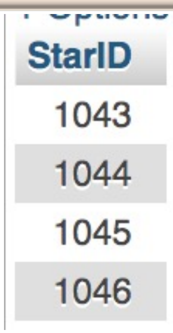
```
SELECT  distinct S1.StarID
FROM     Movie M1, StarsIn S1,
         Movie M2, StarsIn S2
WHERE
         M1.MovieID = S1.MovieID AND M1.year = 1944 AND
         M2.MovieID = S2.MovieID AND M2.year = 1974 AND
        S2.StarID = S1.StarID
```

Rewriting INTERSECT with Joins

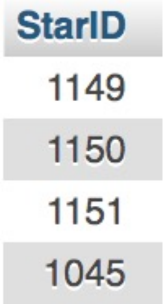
Movie(<u>MovieID</u>, Title, Year)
StarsIn(<u>MovieID, StarID</u>, Role)
MovieStar(<u>StarID</u>, Name, Gender)

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

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



StarID
1043
1044
1045
1046



StarID
1149
1150
1151
1045

```
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
```

Difference in SQL

- EXCEPT(also referred to as MINUS) Produces a relation that includes all the tuples that appear in R1, but do not appear in R2.
  - R1 and R2 must be union compatible.

```
SELECT ...
EXCEPT [ALL] SELECT ...
[EXCEPT [ALL] SELECT ...]
```

EXCEPT Example

- Find IDs of stars who have been in a movie in 1944 but not in 1974.

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

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

EXCEPT is part of the SQL standard, but is not implemented in MySQL.

EXCEPT queries can be implemented with nested queries – stay tuned!

CREATE, ALTER and DROP TABLE statements

Basic SELECT Query

Set Operations

Aggregation, GROUP BY and HAVING

Aggregation in SQL

- Aggregates are functions that produce summary values.

```
SELECT [DISTINCT] (attribute / exprsn / aggregation-function list | * )
FROM <table list>
[WHERE [join condition and]    search_condition]
[ORDER BY column_name [ASC|DESC] {, column-name [ASC|DESC]}];
```

- The aggregation-function list may include:
  - **SUM/ AVG** ([DISTINCT] expression): Calculates the sum/ average of a set of *numeric* values
  - **COUNT** ([DISTINCT] expression): Counts the number of tuples that the query returns
  - **COUNT**(*)
  - **MAX/MIN**(expression): Returns the maximum (minimum) value from a set of values which have a *total ordering*. Note that the domain of values can be non-numeric.

Aggregate Operators Examples

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

# students

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

Finding average GPA of students from high schools with less than 500 students

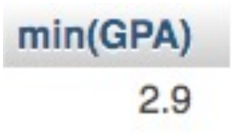
```
SELECT  AVG (GPA)
FROM    Student
WHERE  sizeHS<500
```

Aggregation Examples

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

- Find the minimum GPA.

```
SELECT min(GPA)
FROM Student
```



min(GPA)
2.9

- Find how many students have applied to ‘Stanford’.

```
SELECT count(distinct sID)
FROM Apply
where cname like 'Stanford'
```

Note: want distinct for when Students apply to more than one major at Stanford

GROUP BY and HAVING

- Divide tuples into groups and apply aggregate operations to each group.

- Example: Find the enrollment of the smallest college from each state

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

GROUP BY Syntax

- Aggregation functions can also be applied to groups of rows within a table. The GROUP BY clauses provides this functionality.

```
SELECT [DISTINCT] (attribute / expression / aggregation-function list | * )
FROM <table list>
[WHERE [join condition and]      search_condition]
[GROUP BY grouping attributes]
[ORDER BY column_name [ASC|DESC] {, column-name [ASC|DESC]}];
```

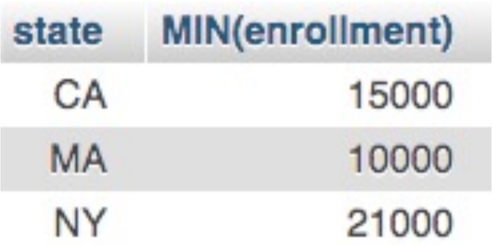
- When GROUP BY is used in an SQL statement, any attribute appeared in SELECT Clause must also appeared in an aggregation function or in GROUP BY clause.

Grouping Examples

- Example: *Find the enrollment of the smallest college from each state*

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

```
SELECT state,  MIN(enrollment)
FROM College
GROUP BY state
```



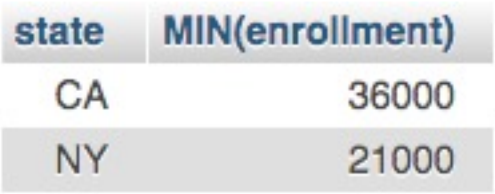
state	MIN(enrollment)
CA	15000
MA	10000
NY	21000

Grouping Examples

- Example: *Find the enrollment of the smallest college from each state which has more than 15000 students.*

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

```
SELECT state,  MIN(enrollment)
FROM College
WHERE enrollment > 15000
GROUP BY state
```



state	MIN(enrollment)
CA	36000
NY	21000

Conditions on Groups

- Conditions can be imposed on the selection of groups to be included in the query result.
- The HAVING clause (following the GROUP BY clause) is used to specify these conditions, similar to the WHERE clause.

```
SELECT [DISTINCT] (attribute / expression / aggregation-function list | * )
FROM <table list>
[WHERE [join condition and] search_condition]
[GROUP BY grouping attributes]
[HAVING <group condition>]
[ORDER BY column_name [ASC|DESC] {, column-name [ASC|DESC]}];
```

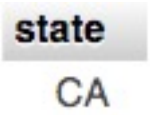
- Unlike the WHERE clause, the HAVING clause can also include aggregates.

Grouping Examples with Having

- *Find states that have more than one college.*

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

```
SELECT      state
FROM        College
GROUP BY   state
HAVING  COUNT(*) > 1
```



state
CA

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.*
3. partition the remaining tuples into groups by where 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.

Clicker Question on 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
```

Which of the following is in the result?

A. (1,3,2)

B. (4,2,6)

C. (4,3,1)

D. All of the above

E. None of the above

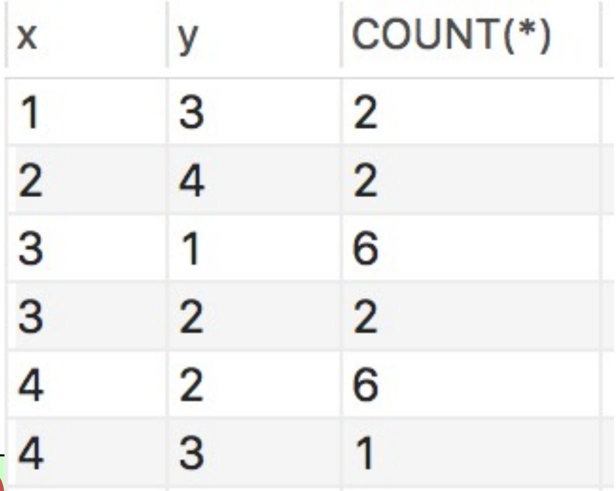
x	y
1	2
1	2
2	3
3	4
3	4
4	1
4	1
4	1
4	2

Tip: You can think of Arc as being a flight, and the query as asking for how many ways you can take each 2 hop plane trip clickergrouping.sql

Clicker Question on 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	COUNT(*)
1	3	2
2	4	2
3	1	6
3	2	2
4	2	6
4	3	1

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

Which is in the result?

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

Tip: You can think of Arc as being a flight, and the query as asking for how many ways you can take each 2 hop plane trip

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

Correct. A cannot appear unaggregated

B. B

C. Count (B)

D. All can appear

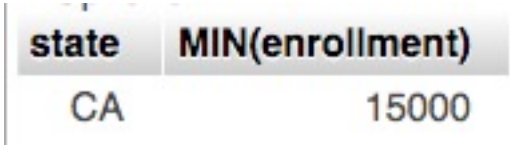
E. None can appear

Grouping Examples

- Find the enrollment of the smallest college with enrollment >10000 for each state with at least 2 colleges (of enrollment >10000)

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

```
SELECT  state,  MIN(enrollment)
FROM    College
WHERE   enrollment > 10000
GROUP BY  state
HAVING  count(*)  > 1
```



```
state   MIN(enrollment)
 CA              15000
```