INFS1200/7900
Introduction to Information Systems

Structured Query Language (SQL)

Hassan Khosravi

Learning Objectives

Description	Tag
Create basic SQL queries using: SELECT, FROM, WHERE statements.	SQL-basic
Create SQL queries containing the DISTINCT statement.	
Create SQL queries using set operators.	
Create SQL queries using aggregate operators.	
Create SQL queries containing GROUP BY statements.	
Create SQL queries containing HAVING statements.	
Given a SQL query and table schemas and instances, compute the query result.	
Create nested SQL queries.	SQL-advance
Create SQL Queries that use the division operator.	
Explain the purpose of NULL values, and justify their use. Also describe the difficulties added by having NULLs.	
Create SQL queries that use joins.	
Create SQL queries that use the CHECK statement.	
Create SQL queries that use ASSERTIONS.	
Create, use and modify VIEWs in SQL.	
Modify data stored in a database using the INSERT, DELETE, and UPDATE statements.	
Identify the pros and cons of using general table constraints (e.g., ASSERTION, CHECK) and triggers in databases.	
Show that there are alternative ways of coding SQL queries to yield the same result.	
Determine whether or not two SQL queries are equivalent.	
Create SQL queries for creating tables.	SQL-DDL
Create SQL queries for altering tables.	
Create SQL queries to enforce referential integrities.	
Create SQL queries for dropping tables.	

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

MySQL Workbench: an open source visual database design tool that integrates SQL development, administration, database design, creation and maintenance into a single integrated development environmentfor MySQL.

Databases used for providing examples: Available for download from the course website.

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

Movie Example Instance

Movie:

MovieID	Title	Year
1	Star Wars	1977
2	Gone with the Wind	1939
3	The Wizard of Oz	1939
4	Indiana Jones and the Raiders of the Lost Ark	1981

StarsIn:

MovieID	StarID	Role
1	1	Han Solo
4	1	Indiana Jones
2	2	Scarlett O’Hara
3	3	Dorothy Gale

MovieStar:

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

SQL Statements

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

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

CREATE, ALTER and DROP TABLE statements

Basic SELECT Query

Set Operations

Aggregation, GROUP BY and HAVING

Nested Queries

Views

Null Values and Joins

INSERT, DELETE and UPDATE statements

Constraints and Triggers

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>} ] )
```

Interpreting the syntax*

KEYWORD

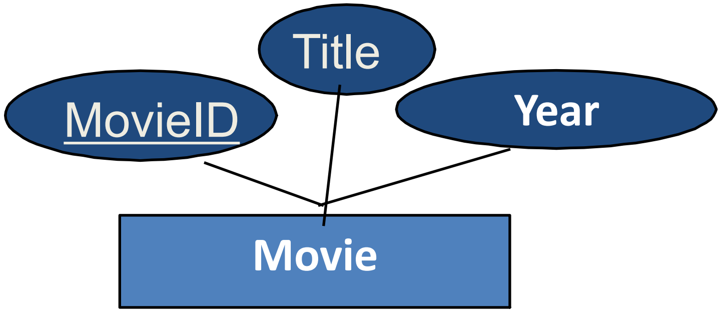
<argument>

[optional]

{multiple}

…|..choice..|…

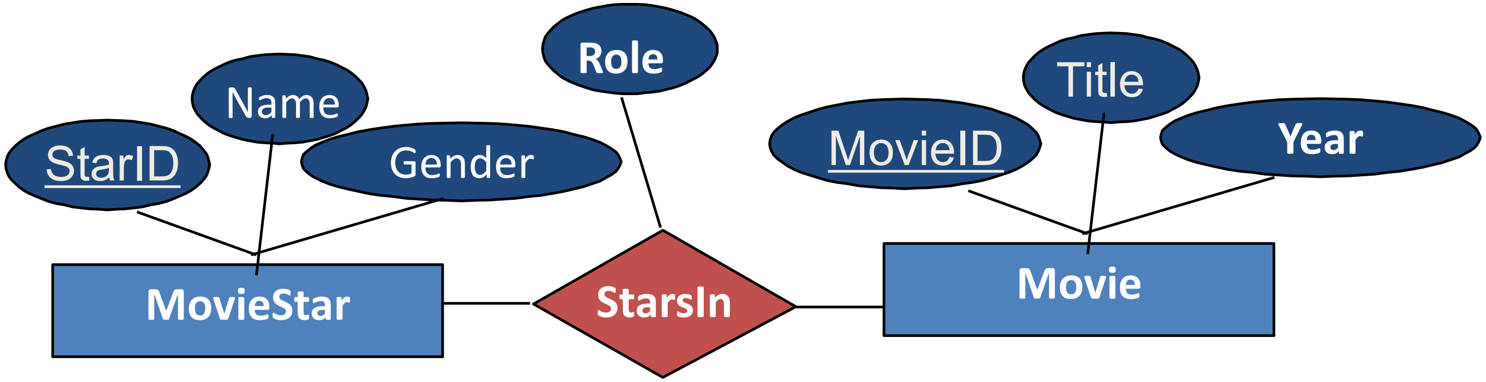
Creating an Entity Table



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,
  FOREIGN KEY (MovieID)  REFERENCES Movies)
```

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 WorkOn tuples that refer to it to null?
  - Set MID in WorkOn tuples that refer to it to default value?

Enforcing Referential Integrity

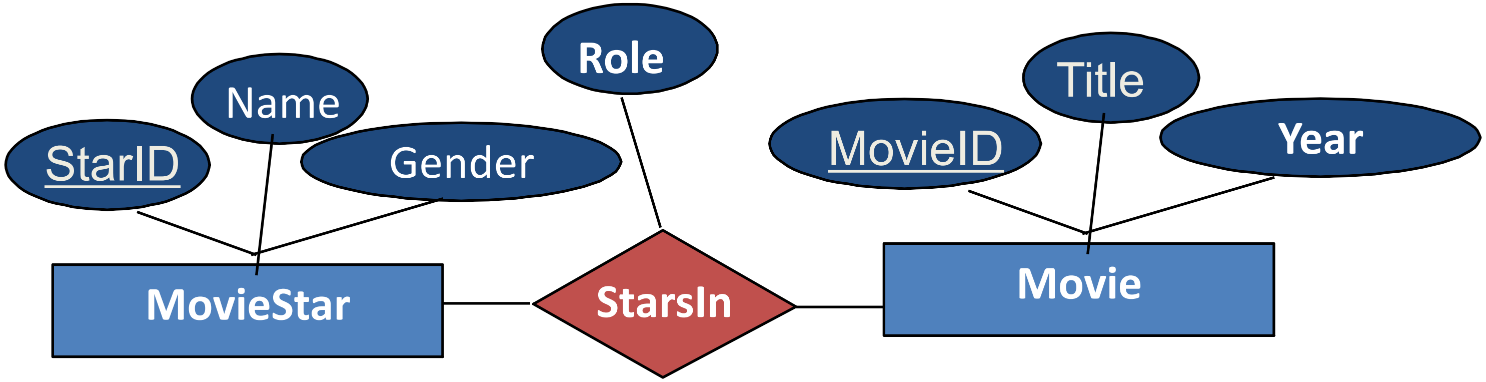
- A **referential triggered action** clause can be attached to a foreign key constraint, that specifies the action to take if a referenced tuple is deleted, or a referenced primary key value is modified.
- By default no action is taken and the delete/update is rejected.

- 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,
      ON DELETE CASCADE
      ON UPDATE CASCADE
  FOREIGN KEY (MovieID)  REFERENCES Movies)
       ON DELETE CASCADE
       ON UPDATE CASCADE
```

Clicker Question

Consider the following table definition.

```
CREATE TABLE  BMW  ( bid  INTEGER, sid INTEGER,  …
                PRIMARY KEY (bid),
                FOREIGN KEY (sid) REFERENCES STUDENTS
                             ON DELETE CASCADE);
```

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

A. If the row for sid value 5678 in STUDENTS is deleted, then the row with bid = 1000 in BMW is automatically deleted.

B. If a row with sid value 5678 in BMW is deleted, then the row with sid=5678 in STUDENTS is automatically deleted.

C. Both of the above.

Clicker Question

Consider the following table definition.

```
CREATE TABLE  BMW  ( bid  INTEGER, sid INTEGER,  …
                PRIMARY KEY (bid),
                FOREIGN KEY (sid) REFERENCES STUDENTS
                             ON DELETE CASCADE);
```

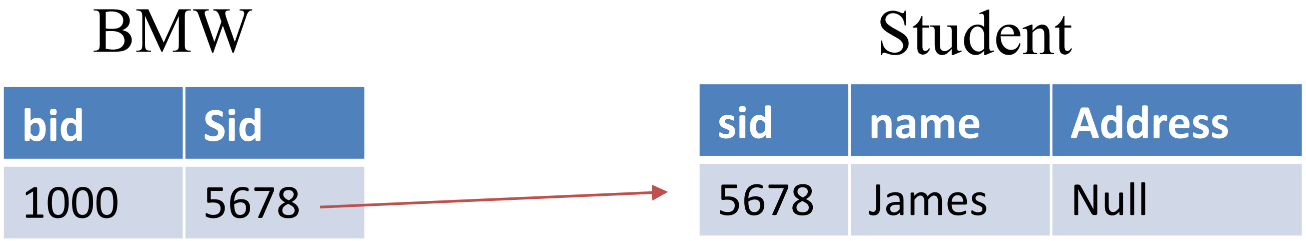
If bid = 1000 and sid = 5678 for a row in Table BMW, choose the best answer

A. If the row for sid value 5678 in STUDENTS is deleted, then the row with bid = 1000 in BMW is automatically deleted.

A is the correct answer

B. If a row with sid value 5678 in BMW is deleted, then the row with sid=5678 in STUDENTS is automatically deleted.

C. Both of the above.



BMW

bid	Sid
1000	5678

Student

sid	name	Address
5678	James	Null

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

Nested Queries

Views

Null Values and Joins

INSERT, DELETE and UPDATE statements

Constraints and Triggers

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, which are bags or multisets and not sets. 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
```

Results

Title
Star Wars
Gone with the Wind
The Wizard of Oz
Indiana Jones and the Raiders of the Lost Ark

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
```

Results

Year
1977
1939
1939
1981

Query

```
SELECT DISTINCT Year
FROM       Movie
```

Results

Year
1977
1939
1981

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. 1 appears once
B. 5 appears twice
C. 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
```

Results

Year
1977
1939
1939
1981

Query

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

Results

Year
1979
1941
1941
1983

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   *
FROM     MovieStar
WHERE    Gender = "Male"
```

Results

StarID	Name	Gender
1	Harrison Ford	Male

Clicker Question: Selection

- Consider the given table and SQL query.

```
SELECT *
FROM Scores
WHERE RunsFor > 5
```

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 one of the following tuple is in the result?

A. (Swallows, Carp, 6, 4)

B. (Swallows, Carp, 4)

C. (12)

D. (*)

Clicker Question: Selection

- Consider the given table and SQL query.

clickerselection.sql

```
SELECT *
FROM Scores
WHERE RunsFor > 5
```

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 one of the following tuple is in the result?

A. (Swallows, Carp, 6, 4) Correct

B. (Swallows, Carp, 4)

C. (12)

D. (*)

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 BETWEEN10000 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 been in a movie.

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

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

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

The row 1, Harrison Ford will appear twice

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

Week 9 Lecture

Join Example

- Find the ids, names and characters of all movie stars who have been in the movie with MovieID 10

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 = 10
```

Results

StarID	Name	Role
1	Harrison Ford	Han Solo

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"
```

Results

StarID	Name	Role
2	Scarlett O'Hara	Scarlett O'Hara

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

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

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

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]}];
```

for the similar things in the first column, ordered by the second column

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)

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

Clicker question: sorting

- Consider the following query:

```
SELECT a, b, c
FROM R
ORDER BY c DESC, b ASC;
```

the tuple before tuple(5,5,5)

- 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) correct

D. All of the above

E. None of the above

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

Nested Queries

Views

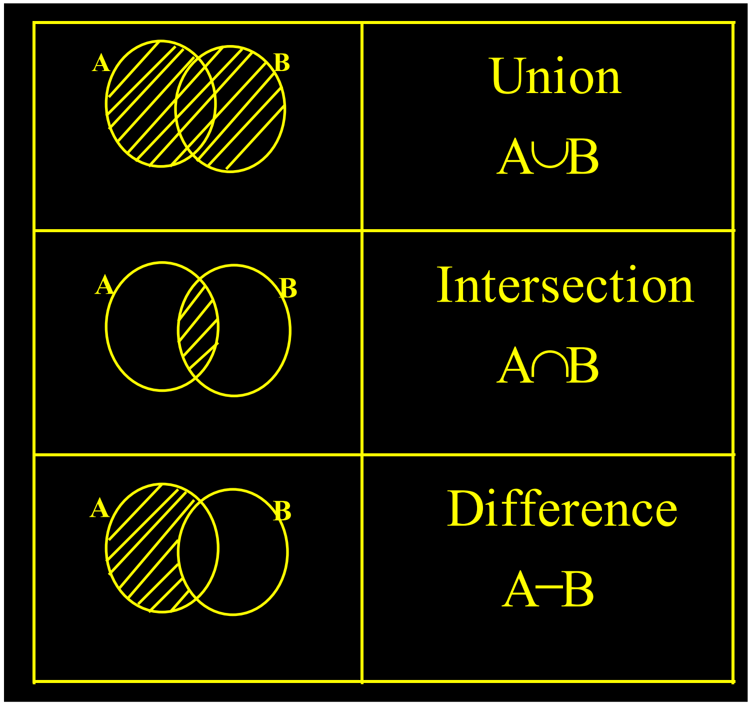
Null Values and Joins

INSERT, DELETE and UPDATE statements

Constraints and Triggers

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)
```

will keep the duplicates

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

will remove the duplicates

```
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)

Need to use intersect

BUT intersect is not part of SQL

```
~~SELECT StarID~~
~~FROM MovieStar MS, StarsIn S, Movie M~~
~~WHERE MS.StarID = S.StarID AND M.MovieID = S.MovieID~~
~~          AND (Year = 1944 AND Year = 1974)~~
```

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
```

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.

EXCEPT is part of SQL BUT not part of MySQL

EXCEPT queries can be implemented using nested queries

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

Properties of Set Operators

$A \cup B$	commutative	$A \cup B = B \cup A$
	associative	$(A \cup B) \cup C = A \cup (B \cup C)$
$A \cap B$	commutative	$A \cap B = B \cap A$
	associative	$(A \cap B) \cap C = A \cap (B \cap C)$
$A - B$	not commutative	$A - B \neq B - A$
	not associative	$(A - B) - C \neq A - (B - C)$

CREATE, ALTER and DROP TABLE statements

Basic SELECT Query

Set Operations

Aggregation, GROUP BY and HAVING

Nested Queries

Views

Null Values and Joins

INSERT, DELETE and UPDATE statements

Constraints and Triggers

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

# students

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

Find name of the oldest student(s)

Finding average age of SR students

Aggregation Examples

- Find the minimum student age.

```
SELECT MIN(age)
FROM Student
```

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

```
SELECT COUNT(distinct SID)
FROM Apply
WHERE cname like 'Stanford'
```

exclude the situation where student who sent twice application to the same college

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

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

- 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
```

based on tuple before grouping

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
…	…

No Animal Science

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

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

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 attributes in *grouping-list*.

   where
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
```

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

Which of the following is in the result?

- (1,3,2) (1,2)(2,3), (1,2)(2,3)

A. (4,2,6)

B. (4,3,1)

[✓ C. All of the above] all of above are correct

D. None of the above

e.g. transfer flight, the first number is the start, the second number is destination
the third number is the number of ways from start to destination

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] correct group by didn't contain A

B. B

C. Count (B)

D. All can appear

E. None can appear

Grouping Examples

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

group by major

Student (_Snum_, Major, Age)

```
SELECT Major, MIN(Age)
FROM Student
WHERE Age>18
GROUP BY Major
HAVING COUNT(*)>1
```

CREATE, ALTER and DROP TABLE statements

Basic SELECT Query

Set Operations

Aggregation, GROUP BY and HAVING

Nested Queries

Views

Null Values and Joins

INSERT, DELETE and UPDATE statements

Constraints and Triggers

Motivating Example for Nested Queries

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

```
SELECT  Distinct 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:

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

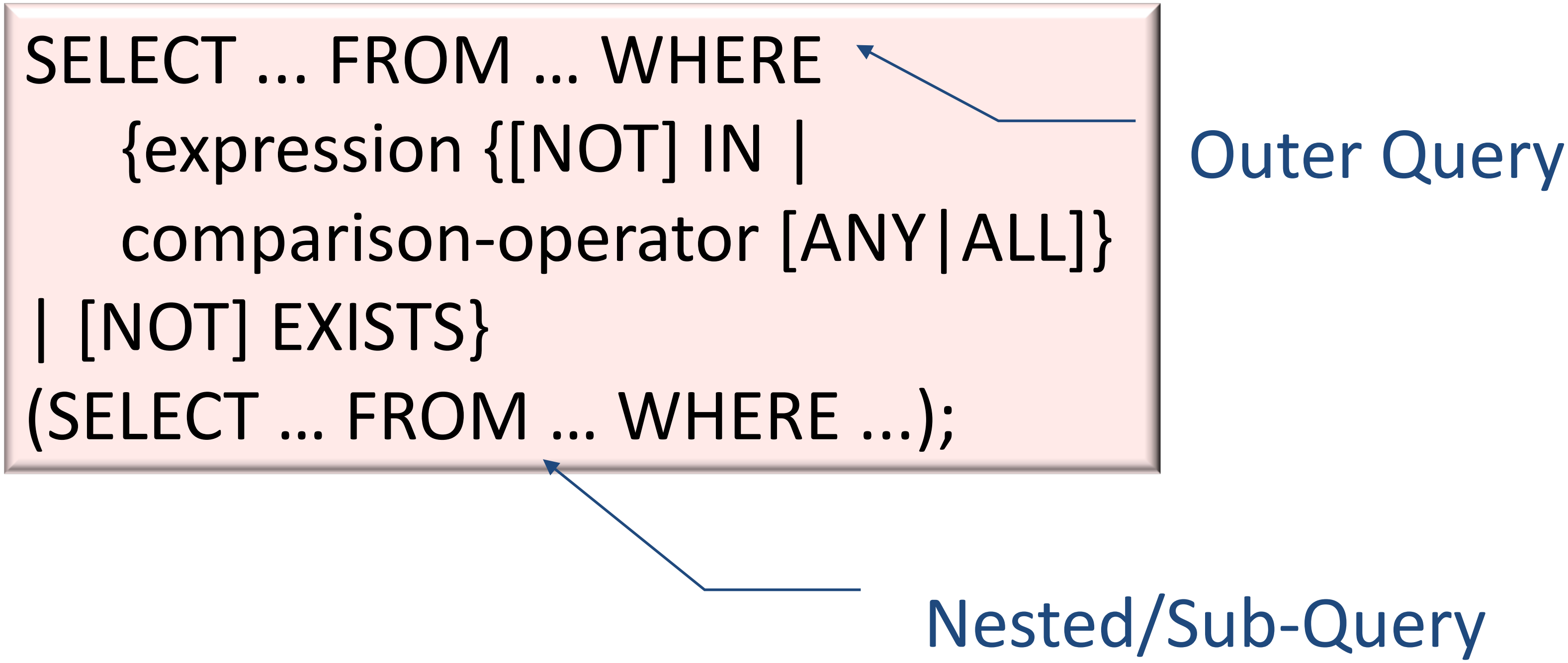
Nested Queries in SQL

- Concept of nested sub-queries
- Correlated and non-correlated variants
- Sub-query operators
- Sub-queries vs. set operations and joins
- Division in SQL

Nested SQL Queries

- A nested query (often termed sub-query) is a query that appears within another query.
  - Inside the WHERE clause of another SELECT statement.
  - Inside an INSERT, UPDATE or DELETE statement.
  - Nesting can occur at multiple levels.
- Nested queries are useful for expressing queries where data must be fetched and used in a comparison condition.

Syntax of Nested SQL Queries



```
SELECT … FROM … WHERE
    {expression {[NOT] IN |
    comparison-operator [ANY|ALL]}
| [NOT] EXISTS}
(SELECT … FROM … WHERE …);
```

- Other query search conditions (including joins) can also appear in the outer query WHERE clause, either before or after the inner query

Nested Queries in SQL

- Concept of nested sub-queries
- Correlated and non-correlated variants
- Sub-query operators
- Sub-queries vs. set operations and joins
- Division in SQL

Non-correlated Nested Queries

- Non-correlated Nested Queries
  - Results are returned from an inner query to an outer clause, that is sub-queries are evaluated from the “inside out”.
  - The outer query takes an action based on the results of the inner query.

run faster

Non-correlated Nested Queries Example

- 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

- In this example, the 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
```

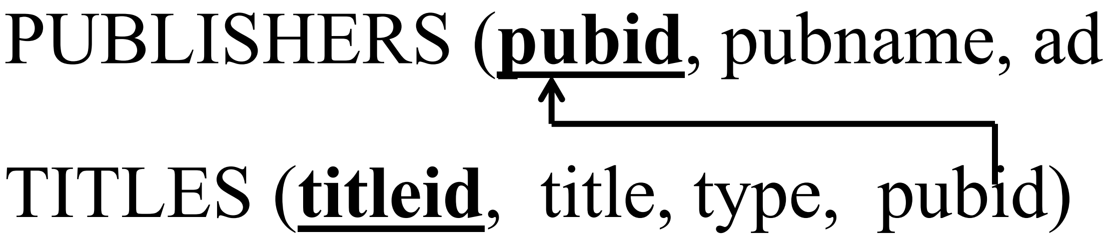
StarID
1026
1027

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

Correlated Nested Queries

- Correlated Nested Queries
  - Correlated subqueries have conditions in their WHERE clause that references some attribute of a relation declared in the outer query.
  - The outer SQL statement provides the values for the inner subquery to use in its evaluation.
  - The subquery is evaluated once for each (combination of) tuple in the outer query.

Correlated vs Non-Correlated Queries



PUBLISHERS (**pubid**, pubname, address, city, state)

TITLES (**titleid**, title, type, pubid)

- Example: Find the name of publishers who publish business books

```
SELECT pubname
FROM publishers p
WHERE 'business' in
    (SELECT type FROM titles
    WHERE
    pubid = p.pubid)
```

Correlated

```
SELECT pubname
FROM publishers
WHERE pubid in
    (SELECT pubid FROM titles
    WHERE
    type = 'business')
```

Non-Correlated

For m=5 publishers and n=100 books, there are m x n =500 scans in correlated queries but only m + n = 105 scans in non-correlated queries.

Nested Queries in SQL

- Concept of nested sub-queries
- Correlated and non-correlated variants
- Sub-query operators
- Sub-queries vs. set operations and joins
- Division in SQL

Sub-query Operators

- Sub-queries that return a set
  - – expression {[NOT] IN (*sub-query*)
  - – expression comp-op [ANY|ALL] (*sub-query*)

- Subqueries that return a single value
  - – expression comp-op (sub-query)

Sub-query returning a Set

Expression and attribute list in sub-query SELECT clause must have same domain

- expression {[NOT] IN (*sub-query*)
  - expression is checked for **membership** in the set (of tuples) returned by sub-query.
- expression comp-op [ANY|ALL] (*sub-query*)
  - expression is **compared** with the set (of tuples) returned by the sub-query
  - ANY: Evaluates to true if one comparison is true
  - ALL: Evaluates to true if all comparisons are true

IN/NOT IN Operator Example

- 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

- In this example, the 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)
```

ANY/ ALL Operator Example

- 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

- Assuming we have multiple movies named Fargo, how would the use of ALL vs. ANY affect the result?

Equivalence of IN and = ANY

- 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)
```

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

Equivalence of Not IN and <>ALL

- Find ids and names of stars who have NOT been in movie with ID 28:

```
SELECT  M.StarID, M.Name
FROM    MovieStar M
WHERE   M.StarID NOT IN  (SELECT  S.StarID
                          FROM    StarsIn S
                          WHERE   MovieID=28)
```

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

Non-Equivalence of Not IN and <>ANY

- If a sub-query returns: {$30K, $32K, $37K}

- NOT IN means
  - NOT=$30K AND NOT=$32K AND NOT=$37K

- <> ANY means
  - NOT=$30K *OR* NOT=$32K *OR* NOT=$37K

<> ANY will be forever true for any value, try value $50K, which will be true for the <>ANY.

Example

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

Clicker Nested Question

Consider the following table and SQL query:

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

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

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

Nested Grouping Examples

Snum	Major	Age

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

- 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.

Nested Grouping Examples

Snum	Major	Age

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

Rules of sub-query construction

- Sub-query Rule 1: Sub-query returning a Set
  - The select list of an inner sub-query introduced with a comparison operator (and ANY/ALL) or IN can include only one expression or column name.
  - The expression you name in the WHERE clause of the outer statement must be join compatible with the column you name in the sub-query select list.

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

Sub-query returning a Value

Expression and attribute list in sub-query SELECT clause must have same domain.

- **expression comp-op** (sub-query)
  - expression is **compared** with the *value* returned by the sub-query.
  - The sub-query must evaluate to a single value otherwise an error will occur.
  - You must be familiar enough with your data and application to forecast this.

Single Value Subquery Example

- Find the name of the most recent movies.

```
SELECT Title
FROM Movie
Where year = (SELECT max(year)
                FROM Movie)
```

- Aggregate functions always guarantee a single value return

Using the Exists Function

- The EXISTS function tests for the existence or nonexistence of data that meet the criteria of the sub-query

```
SELECT … FROM …
WHERE [NOT] EXISTS (sub-query)
```

- Sub-queries are used with EXISTS and NOT EXISTS are always correlated
  - WHERE EXISTS (sub-query) evaluates to true if the result of the correlated sub-query is a non-empty set, i.e. contains 1 or more tuples.
  - WHERE NOT EXISTS (sub-query) evaluates to true if the result of the correlated sub-query returns an empty set, i.e. zero tuples.

Exists NOT EXISTS Example

- Find movies that were the only movie of the year.

```
SELECT  *
From Movie M1
where NOT EXISTS
     (Select *
      FROM Movie M2
      Where M1.movieid <> M2.movieID and M1.year = M2.year)
```

- Find movies that were not the only movie of the year.

```
SELECT  *
From Movie M1
where EXISTS
     (Select *
      FROM Movie M2
      Where M1.movieid <> M2.movieID and M1.year = M2.year)
```

EXISTS Example

- For each college, check if there is another college 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

- Illustrates why, in general, sub-query must be re-computed for each college tuple.

Rules of sub-query construction

- Sub-query Rule 2: Using the Exists Function
  - The select list of a sub-query introduced with EXISTS almost always consists of the asterisk (*).
  - However, they are no syntactic restrictions on specifying a select list for a sub-query introduced by EXISTS

```
SELECT  *
From Movie M1
where EXISTS
     (Select *
      FROM Movie M2
      Where M1.movieid <> M2.movieID and M1.year = M2.year)
```

Rules of sub-query construction

- Sub-query Rule 3
  - Sub-queries cannot include the ORDER BY clause. The optional DISTINCT keyword may effectively order the results of a sub-query, since most systems eliminate duplicates by first ordering the results

Nested Queries in SQL

- Concept of nested sub-queries
- Correlated and non-correlated variants
- Sub-query operators
- Sub-queries vs. set operations and joins
- Division in SQL

Sub-queries vs. Set Operations and Joins

- Find IDs of stars who have been in movies in 1944 and 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
```

```
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
```

```
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)
```

Sub-queries vs. Set Operations and Joins

- Find IDs of stars who have been in movies in 1944 and 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
```

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

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

Sub-queries vs. Set Operations in Difference

- Find IDs of stars who have been in movies in 1944 and not in1974.

```
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
```

```
SELECT S.StarID
FROM    Movie M, StarsIn S
WHERE  M.MovieID = S.MovieID AND M.year = 1944 AND
 S.StarID NOT IN (SELECT  S2.StarID
              FROM Movie M2, StarsIn S2
              WHERE   M2.MovieID = S2.MovieID AND M2.year = 1974)
```

Nested Queries Example

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

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

Nested Query Example (Tricky)

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

Why are Duplicates Important?

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

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

Joins vs Sub-queries

- Many nested queries are equivalent to a simple query using JOIN operation. However, in many cases, the use of nested queries is necessary and cannot be replaced by a JOIN operation.
-
- Advantages of join queries
  - The join implementation can display data from all tables in the FROM clause whereas, the sub-query implementation can only display data from the table(s) in the outer query.
- Advantages of nested sub-queries
  - The ability to calculate an aggregate value on the fly and feed it back to the outer query for comparison is an advantage sub-queries have over joins.
- Conclusion
  - Use joins when you are displaying results from multiple tables.
  - Use sub-queries when you need to compare aggregates to other values.

Nested Queries in SQL

- Concept of nested sub-queries
- Correlated and non-correlated variants
- Sub-query operators
- Sub-queries vs. set operations and joins
- Division in SQL

Division in SQL

- Division in SQL is useful for answering queries include a “**for all**” or “**for every**” phrase, e.g., Find movie stars who were in **all** movies.

- Unfortunately, there is no direct way to express division in SQL. We can write this query, but to do so, we will have to express our query through double negation and existential quantifiers.

Examples of Division A/B

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

A/B1

sno
s1
s2
s3
s4

A/B2

sno
s1
s4

A/B3

sno
s1

Division in SQL Using EXCEPT

- Find the IDs of movie stars who have played in all of the movies.

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

```
SELECT StarID
FROM   MovieStar MS
WHERE NOT EXISTS
All movies    ((SELECT  MovieID
                 FROM  Movies)
Movies          EXCEPT
Played by MS    (SELECT  MovieID
                FROM  StarsIn S
                WHERE  S.StarID=MS.StarID))
```

Division in SQL Using NOT EXISTS

- Find the IDs of movie stars who have played in all of the movies.

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

select Movie Star MS such that
there is no Movie M…
which is not played by MS

```
SELECT StarID
FROM    MovieStar  MS
WHERE NOT EXISTS
        (SELECT  M.MovieID
        FROM  Movie M
        WHERE  NOT EXISTS
          (SELECT  S.MovieID
               FROM  StarsIn  S
              WHERE  S.MovieID=M.MovieID
              AND S.StarID=MS.StarID))
```

CREATE, ALTER and DROP TABLE statements

Basic SELECT Query

Set Operations

Aggregation, GROUP BY and HAVING

Nested Queries

Views

Null Values and Joins

INSERT, DELETE and UPDATE statements

Constraints and Triggers

Motivating Example for Use of Views

- 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

- A Better alternative is to use views

What Are Views?

- A View is a single table that is derived from other tables, which could be base tables or previously defined views
- Views can be
  - Virtual tables - that do not physically exist on disk.
  - Materialized - by physically creating the view table. These must be updated when the base tables are updated
- We can think of a virtual views as a way of specifying a table that we need to reference frequently, even though it does not physically exist.

Benefits of Using Views

- **Simplification**: View can hide the complexity of underlying tables to the end-users.
- **Security**: Views can hide columns containing sensitive data from certain groups of users.
- **Computed columns**: Views can create computed columns, which are computed on the fly.
- **Logical Data Independence**: Views provide support for logical data independence, that is users and user's programs that access the database are immune from changes in the logical structure of the database.

Defining and Using Views

```
CREATE VIEW <view name>
      (<column name> {, <column name>}) AS
       <select statement> ;
```

- Example: Suppose we have the following 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).

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

- 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.

View Updates

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

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

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

Example: UQ 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.

Dropping Views

```
DROP VIEW [IF EXISTS]
view_name [, view_name] ...
 [RESTRICT | CASCADE]
```

- Dropping a view does not affect any tuples of the underlying relation.

- DROP TABLE command has options to prevent a table from being dropped if views are defined on it:
  - RESTRICT : drops the table, unless there is a view on it
  - 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.

```
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

```
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)
```

Clicker Question on Views

Consider the following table and SQL queries:

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

```
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

CREATE, ALTER and DROP TABLE statements

Basic SELECT Query

Set Operations

Aggregation, GROUP BY and HAVING

Nested Queries

Views

Null Values and Joins

INSERT, DELETE and UPDATE statements

Constraints and Triggers

NULL Values

- A value of NULL indicates that the value is unknown.
- The predicate **IS NULL** (**IS NOT NULL**) can be used to check for null values.
- Example: Find all student names whose age is not known.

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

Operations on NULL Values

- 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*

- Comparisons between two null values, or between a NULL and any other value, return unknown because the value of each NULL is unknown.
  - *E.g. 5 < null or null <> null or null = null*

- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes.

```
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

Joins in SQL

- A Join used to combine related tuples from two relations into a single tuple in a new (result) relation.

- Join operation is needed for organizing a search space of data. This is needed when information is contained in more than one relation.

- Join relations are specified in the FROM Clause. When two relations are combined for a search, we need to know how the relations are combined.

- Based on Cartesian Product (denoted as X) There are three types of Join operations:
  - Theta-Join
  - Equi-Join
  - Natural Join

Cartesian Product in SQL

- Every row in R1 is matched with every row in R2 to form tuples in the result relation. The schema of the result relation contains all the columns from R1 and all the columns from R2

MovieStar

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

StarsIn

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

```
SELECT *FROM
MovieStar, StarsIN
```

For |R1| = m, |R2| = n, the |R1 X R2| = m * n

MovieStar x StarsIn

StarID1	Name	Gender	MovieID	StarID2	Character
1	Harrison Ford	Male	1	1	Han Solo
2	Vivian Leigh	Female	1	1	Han Solo
3	Judy Garland	Female	1	1	Han Solo
1	Harrison Ford	Male	4	1	Indiana Jones
…	…	…	…	…	…

Theta-Join in SQL

- Theta-join is the most general type of join, which allows several logical operators {=, ≠, <, ≤, >, ≥}.

- Example: For each student, list the students who are older than him/her (i.e., the first student).

```
SELECT A.EName, B.EName
FROM Student A, Student B
WHERE  A.age < B.age
```

Theta-Join Example

MovieStar

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

StarsIn

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

```
SELECT *
FROM MovieStar M, StarsIN S
Where M.StarID < S.StarID
```

1	Name	Gender	MovieID	5	Character
1	Harrison Ford	Male	2	2	Scarlett O’Hara
1	Harrison Ford	Male	3	3	Dorothy Gale
2	Vivian Leigh	Female	3	3	Dorothy Gale

Equi-Join and Natural Join

- *Equi-Join*: A special case of Theta join where condition contains only ***equalities.***
- 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.

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

Natural Join Examples

MovieStar

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

StarsIn

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

```
Select *
From MovieStar natural join StarsIN
```

StarID	Name	Gender	MovieID	Character
1	Harrison Ford	Male	1	Han Solo
1	Harrison Ford	Male	4	Indiana Jones
3	Judy Garland	Female	3	Dorothy Gale
2	Vivian Leigh	Female	2	Scarlett O’Hara

Inner and Outer Joins

- **Inner Join**

  This is the default join, in which a tuple is included in the result relation, only if matching tuples exist in both relations.

- **Outer Join**

  Outer joins can include the tuples that do not satisfy the join condition, i.e, a matching tuple does not exist, which is indicated by a NULL value

  – Full Outer Join: Includes all rows from both tables.

  – Left Outer Join: Includes all rows from first table.

  – Right Outer Join: Includes all rows from second table .

```
SELECT <attribute list>
FROM table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference ON
< search_condition>
```

Inner and Outer 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 the following relations 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)

A	B
1	2
3	4
5	6

S(B,C,D)

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 with NULLs 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

CREATE, ALTER and DROP TABLE statements

Basic SELECT Query

Set Operations

Aggregation, GROUP BY and HAVING

Nested Queries

Views

Null Values and Joins

INSERT, DELETE and UPDATE statements

Constraints and Triggers

INSERT Statement

- INSERT statement is used to add tuples to an existing relation
- Single Tuple INSERT
  - Specify the relation name and a list of values for the tuple
  - Values are listed in the same order as the attributes were specified in the CREATE TABLE command
  - User may specify explicit attribute names that correspond to the values provided in the insert statement. The attributes not included cannot have the NOT NULL constraint
- Multiple Tuple INSERT
  - By separating each tuple's list of values with commas
  - By loading the result of a query

Single Tuple INSERT Example

```
INSERT INTO <table name>
      [(<column name> {, <column name> })]
(VALUES (<constant value>, {,<constant value> })
            | <select statement>);
```

- Can insert a single tuple using:

```
INSERT  INTO  Student
VALUES  (53688, 'Smith', 3.2, 200)
```

- or

```
INSERT  INTO  Student (sID, sName, GPA, sizeHS)
VALUES  (53688, 'Smith', 3.2, 200)
```

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

```
INSERT  INTO  Student (sID, sName, GPA, sizeHS)
VALUES  (53688, 'Smith', 3.2, NULL)
```

Multiple Tuple INSERT Example

```
INSERT INTO <table name>
      [(<column name> {, <column name> })]
(VALUES (<constant value>, {,<constant value> })
                  | <select statement>);
```

- Can add values selected from another table
- Make student 123 apply into all “BIO” related majors at Stanford.

```
INSERT  INTO  apply
     SELECT  123, "Stanford",  major, NULL
      FROM apply
        WHERE major LIKE "%bio%"
```

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

DELETE Statement

- DELETE statement is used to remove existing tuples from a relation.
- A single DELETE statement may delete zero, one, several or all tuples from a table.
- Tuples are explicitly deleted from a single table.
- Deletion may propagate to other tables if referential triggered actions are specified in the referential integrity constraints of the CREATE (ALTER) TABLE statement

```
DELETE FROM <table name>
     [WHERE <select condition>];
```

DELETE Statement Example

- Delete all “BIO” related applications of student 123 for Stanford.

```
DELETE FROM  apply
WHERE sID = 123 AND
cName LIKE "Stanford" AND major LIKE "%BIO%"
```

- Note that only whole tuples are deleted.

- Can delete all tuples satisfying some condition

UPDATE Statement

UPDATE statement is used to modify attribute values of one or more selected tuples in a relation.

- Tuples are selected for update from a single table.
- However, updating a primary key value may propagate to other tables if referential triggered actions are specified in the referential integrity constraints of the CREATE (ALTER) TABLE statement.

```
UPDATE <table name>
    SET <column name> = <value expression>
        {, <column name>  = <value expression>}
    [WHERE <select condition>];
```

UPDATE Statement Example

- 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?

CREATE, ALTER and DROP TABLE statements

Basic SELECT Query

Set Operations

Aggregation, GROUP BY and HAVING

Nested Queries

Views

Null Values and Joins

INSERT, DELETE and UPDATE statements

Constraints and Triggers

Semantic Constraints

- Keys, entity constraints and referential integrity are structural constraints that are managed by the DBMS.
- Semantic constraints can be specified using CHECK and ASSERTION statements.
- The constraint is satisfied by a database state if no combination of tuples in the database state violates the constraint.

Semantic Constraints: Check

- Semantic constraints over a single table are specified using tCheck conditional-expressions

```
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

Constraints Over Multiple Relations

- Constraints that 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)));
```

Useful for project
Not tested on exams

Triggers

- An active database is a database that includes an event-driven architecture. Triggers are a procedure that start automatically if specified changes occur to the DBMS.

- 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)

- Database vendors did not wait for trigger standards! So trigger format depends on the DBMS

NOTE: triggers may cause cascading effects.

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

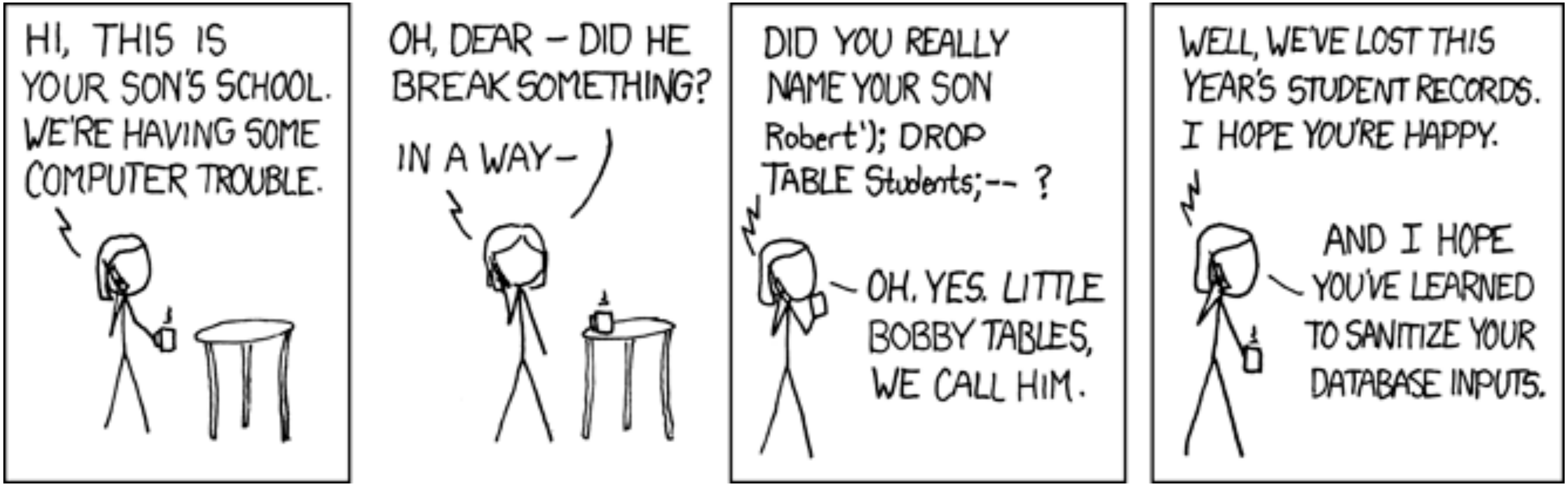
apply once per statement

action

Can be either before or after

And now a brief digression

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


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Learning Objectives Revisited

Description	Tag
Create basic SQL queries using: SELECT, FROM, WHERE statements.	SQL-basic
Create SQL queries containing the DISTINCT statement.	
Create SQL queries using set operators.	
Create SQL queries using aggregate operators.	
Create SQL queries containing GROUP BY statements.	
Create SQL queries containing HAVING statements.	
Given a SQL query and table schemas and instances, compute the query result.	
Create nested SQL queries.	SQL-advance
Create SQL Queries that use the division operator.	
Explain the purpose of NULL values, and justify their use. Also describe the difficulties added by having NULLs.	
Create SQL queries that use joins.	
Create SQL queries that use the CHECK statement.	
Create SQL queries that use ASSERTIONS.	
Create, use and modify VIEWs in SQL.	
Modify data stored in a database using the INSERT, DELETE, and UPDATE statements.	
Identify the pros and cons of using general table constraints (e.g., ASSERTION, CHECK) and triggers in databases.	
Show that there are alternative ways of coding SQL queries to yield the same result.	
Determine whether or not two SQL queries are equivalent.	
Create SQL queries for creating tables.	SQL-DDL
Create SQL queries for altering tables.	
Create SQL queries to enforce referential integrities.	
Create SQL queries for dropping tables.	