Lecture 3:
SQL1, Basic SQL

Instructor: Shel Finkelstein

Reference:
A First Course in Database Systems,
3rd edition, Chapter 2.3, 6.1

Important Notices

- Lecture slides and other class information will be posted on Piazza (not Canvas).
  - Slides are posted under Resources→Lectures
  - Lecture Capture recordings are available to all students under Yuja.
    - Recording for CSE 180 lecture for Wednesday, January 17 was posted directly on Yuja, since I was at a conference on that date.
  - There's no Lecture Capture for Lab Sections.
- All Lab Sections (and Office Hours) normally meet In-Person, with rare exceptions announced on Piazza.
  - Some slides for Lab Sections will be posted on Piazza under Resources→Lab Section Notes
- Office Hours for TAs and me are posted in Syllabus and Lecture1, as well as in Piazza notice @7.

Important Notices

- Lab1 assignment has been posted on Piazza under Resources→Lab1. General Information about Labs has also been posted Resources→GeneralInformation
  - See Piazza announcement about Lab1, the easiest of the Lab Assignments.
  - Lab1 is due **Tuesday, January 23**, by 11:59pm.
    - Late Lab Assignments will not be accepted.
    - Be sure that you post the correct file!
    - Students who enroll late based on the waitlist won't receive extensions.
    - No students will be be able to enroll from the waitlist after that date.
  - Your solution should be submitted via Canvas as a zip file.
    - Canvas will be used for both Lab submission and grading.
  - You had enough information to do Lab1 after the Wednesday, January 17 recorded Lecture.
  - A load file, load_lab1.sql, has been posted under Resources→Lab1 to help you test your Lab1 solution.
    - But testing can only prove that a solution is incorrect, not that it is correct.
  - Lab1 will be discussed at Lab Sections, helping students with concepts.
    - But each student must complete Lab Assignments individually.

Important Notices

- First Gradiance Assignment, CSE 180 Fall 2023, was posted on Thursday, January 18, and is due (on Gradiance) on **Friday, January 26**, by 11:59pm.
  - The day of the week on which Gradiance Assignments are due will vary.
  - For Gradiance this quarter, the token is **EC7184D3**
  - There are 9 questions in this Gradiance Assignment.
  - You may take each Gradiance Assignment as many time as you like.
    - Questions stay the same, but answers change.
    - Your Gradiance Assignment score is the score on your **Final** Gradiance submission before the deadline.
    - Gradiance does not allow late submissions, so no extensions are possible.
  - Suggest that you use your cruzid (or something similar) as your Gradiance login.
    - Be sure to enter your UCSC email address as your email.
    - If you forget your Gradiance password, you can recover it by providing your Gradiance login and email.
  - After Friday, January 19 Lecture, you should have enough info to complete this Gradiance Assignment.

Group Tutor-1

Alex Asch
alasch@ucsc.edu

Tutoring Times:

- Tuesday: 1:00pm – 2:00pm
- Friday: 9:20am – 10:25am
- Friday: 1:30pm – 2:30pm

Tutoring will be in Jack's Lounge, on the first floor of Baskin Engineering.

Group Tutor-2

Yash Raj Singh
ysingh5@ucsc.edu

Tutoring Times:

- Monday: 100pm – 2:00pm
- Tuesday: 2:00pm – 3:00pm
- Wednesday: 1pm – 2:00pm

Tutoring will be in Jack's Lounge, on the first floor of Baskin Engineering.

Another Practice Homework

(Not collected, will be discussed during next Lecture)

- If D_1 has n_1 elements and D_2 has n_2 elements, then how many elements are there in $D_1 \times D_2$?
- If D_i has n_i elements, then how many elements are there in the Cartesian Product $D_1 \times \ldots \times D_k$?
- If D_1 has n_1 elements and D_2 has n_2 elements, then how many relation instances can one construct from $D_1 \times D_2$?
- If D_i has n_i elements, then how many relation instances can one construct from $D_1 \times \ldots \times D_k$?

Summary of Previous Lecture

- Data model
- Relation Schema
  - Attributes or column names
  - Tuple or row
  - Columns
- Relation Instance
- Relational Database Schema
- A Relational Database Instance (“a database”)
  - An Instance of a Relational Database Schema
- Logical and Physical Data Independence

A Relational Database Schema

- A *relational database schema* or, simply, a *database schema* is a set of relation schemas with disjoint relation names.

- A university database schema:
  - Student(studentID, name, major, gender, avgGPA)
  - Course(courseID, description, department)
  - Teach(profID, courseID, quarter, year)
  - Enroll(studentID, courseID, grade)
  - Professor(profID, name, department, level)

Instance of a Database Schema

- An *instance of a database schema* $\{R_1, \ldots R_k\}$ (or a *database instance* in short) is a set $\{r_1, \ldots, r_k\}$ of relations such that r_i is an instance of R_i, for $1 \leq i \leq k$.

Student

studentID	name	major	gender	avgGPA
112	Ann	Computer Science	F	3.95
327	Bob	Computer Science	M	3.90
835	Carl	Physics	M	4.00

Course

courseID	description	department
CMPS101	Algorithms	CS
BINF223	Intro. to Bio.	Biology

Teach, Enroll, Professor, …

What is a Data Model?

- A *data model* is a mathematical formalism that consists of three parts:
  1. Structure of the data
  2. Operations on the data
  3. Constraints on the data

- This course focuses mainly on the relational data model

- **What is the associated query language commonly used for the relational data model?**

Two Query Languages

- Codd proposed two different query languages for the relational data model.
  - *Relational Algebra*
    - Queries are expressed as a sequence of operations on relations
    - Procedural language
  - *Relational Calculus*
    - Queries are expressed as formulas of first-order logic
    - Declarative language

- **Codd's Theorem**: The Relational Algebra query language has the same *expressive power* as the Relational Calculus query language.

Procedural vs. Declarative Languages

- **Procedural program**
  - The program is specified as a sequence of operations to obtain the desired the outcome. I.e., *how* the outcome is to be obtained.
  - E.g., Java, C, …
- **Declarative program**
  - The program specifies *what* is the expected outcome, and not *how* the outcome is to be obtained.
  - E.g., Scheme, Ocaml, …

An Example: Travel from Baskin Engineering, UC Santa Cruz to Soda Hall, UC Berkeley

Declarative (non-procedural):

- Go from Baskin Engineering, at McLaughlin Dr. and Heller Dr. in Santa Cruz, CA, to Soda Hall, at Hearst Ave. and Oxford St. in Berkeley, CA

An Example: Travel from Baskin Engineering, UC Santa Cruz to Soda Hall, UC Berkeley

Procedural (first part)

- Turn left onto Heller Dr,
- Turn left onto Empire Grade
- Continue onto High St
- High St turns right and becomes Storey St,
- Turn left onto King St
- Turn left onto CA-1 S/Mission St
- Continue to follow CA-1 S
- Take the CA-17 exit on the left toward San Jose/Oakland
- Continue onto CA-17 N
- Keep left to continue on I-880 N

An Example: Travel from Baskin Engineering, UC Santa Cruz to Soda Hall, UC Berkeley

Procedural (continued):

- Keep right at the fork, follow signs for CA-24/I-980/Walnut Cr. Continue onto I-980 E
- Keep left to continue on CA-24 E
- Take the exit toward 51st Street
- Turn right onto 52nd St
- Take the 1st left onto Shattuck Ave
- Turn right onto University Ave
- Turn left onto Oxford St
- Turn right onto Hearst Ave
- Turn left; destination will be on the right

SQL – Structured Query Language

- Is SQL a procedural or a declarative language?
  - SQL is usually described as declarative, but it's not fully declarative.
- SQL is the principal language used to describe and manipulate data stored in relational database systems.
  - Frequently pronounced as “Sequel”, but formally it's “Ess Cue El”
  - Not exactly the same as Codd's relational algebra or relational calculus
- Several iterations of the standard from cooperating groups
  - SQL-86, SQL-89, **SQL-92 (SQL2)**, SQL:1999 (SQL3), SQL:2003, SQL:2006, SQL:2008, SQL:2011
  - ANSI (American National Standards Institute)
  - ISO (International Organization for Standards)
  - Implementations usually offer their own extensions of SQL.

SQL DDL and SQL DML

Two main aspects to SQL:

- *Data Definition Language (DDL)*
  - CREATE TABLE, DROP TABLE
  - CREATE SCHEMA, DROP SCHEMA
  - …
- *Data Manipulation Language (DML)*
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
  - …

Relations in SQL

Two types of Relations:

1. Stored relations (or Tables)
   - These are tables that contain tuples and can be modified or queried.

2. Views
   - Views are relations that are defined in terms of other relations, but they are not stored.
   - They are logical tables, which can be used in queries as if they were tables.

Most of the Primitive Data Types in SQL

- CHAR(n): fixed-length string of up to n characters (which is blank-padded on the right to be exactly n characters)
- VARCHAR(n): string of up to n characters, not blank-padded
- BIT(n)
- BIT VARYING(n):
- BOOLEAN or BOOL: TRUE or FALSE
  - Some systems don't have BOOLEAN and use either BIT(1) or INTEGER, where 0 and 1 correspond to FALSE and TRUE.
- INT or INTEGER
  - Analogous to int in C
- SHORTINT
  - Analogous to short int in C

Primitive Data Types in SQL (continued)

- DECIMAL(n,d), NUMERIC(n,d) – almost the same meaning.
  - Total of n decimal digits; d of them to the right of the decimal point
- FLOAT(p), FLOAT and REAL (Implementation-specific)
- DOUBLE PRECISION
    - Analogous to `double` in C
- DATE, TIME, TIMESTAMP, INTERVAL
  - Separate data types
  - Constants are character strings of a specific form, e.g., DATE '2017-09-13' and TIME '16:45:33'
- A few others …
- PostgreSQL has non-standard TEXT, for variable strings of any length

More on DATE, TIME, TIMESTAMP, INTERVAL

- Some information sources
  - Limited stuff in textbook; see Sections 2.3.2 and 6.1.5
  - Too much in PostgreSQL manual
    - 8.5. Date/Time Types
    - 9.9. Date/Time Functions and Operators
- Some examples of constants
  - DATE '2017-09-13' and TIME '16:45:33'
  - TIMESTAMP '2017-09-13 16:45:33'
  - INTERVAL '2 HOURS 30 MINUTES'
- Arithmetic
  - Subtracting one TIME from another results in an INTERVAL
  - Taking a TIME and adding an INTERVAL results in a TIME
  - Similarly for TIMESTAMP and DATE (instead of TIME)
  - There are functions which extract DATE and TIME from a TIMESTAMP

Use keyword (e.g., DATE) to express the correct Data Type!

Oracle doesn't have the TIME Data Type.

Taking a DATE, plus or minus an INTERVAL, gives a TIMESTAMP

Defining a Table

```
CREATE TABLE Movies (
    movieTitle      CHAR(100),
    movieYear       INT,
    length          INT,
    genre           BOOLEAN,
    studioName      CHAR(30),
    producerC#      INT
);
```

Textbook uses title and year, instead of movieTitle and movieYear.

We also rename some other attributes.

movieTitle	movieYear	length	genre	studioName	producerC#

Think of producerC# *as the Certificate Number for the movie's producer, where Certificate Number is a key uniquely identifying a Movie Executive.*

Defining a Table (continued)

```
CREATE TABLE MovieStar (
    starName        CHAR(30),
    address         VARCHAR(255),
    gender          CHAR(1),
    birthdate       DATE
);
```

starName	address	gender	birthdate

Modifying Relation Schemas

- DROP TABLE Movies;
  - The entire table is removed from the database schema.
- ALTER TABLE MovieStar ADD phone CHAR(16);
  - Adds an attribute “phone” with type CHAR(16) to the table MovieStar.

name	address	gender	birthdate

name	address	gender	birthdate	phone

- ALTER TABLE MovieStar DROP birthdate;

name	address	gender	phone

Default Values

```
CREATE TABLE MovieStar (
    starName        CHAR(30),
    address         VARCHAR(255)  DEFAULT  'Hollywood',
    gender          CHAR(1),
    birthdate       DATE
);
```

- If a new row is inserted and a value is specified for an attribute, then the value for that attribute will be that specified value.
- If a new row is inserted and there's no value specified for an attribute, and there's a DEFAULT value for that attribute, then the value for that attribute will be the DEFAULT.
- If no value is entered for an attribute, and no DEFAULT value is specified for that attribute, and that attribute can be NULL, then the value of the attribute will be NULL.
  - NULL is a special value in SQL used to represent unknown values.
  - Constraints (which we'll discuss soon) may prevent an attribute from having the value NULL.
- If no value is entered for an attribute, and no DEFAULT value is specified for that attribute, and that attribute can't be NULL, then an error is returned.

More Examples of Default Values

```
CREATE TABLE MovieStar (
    starName        CHAR(30),
    address         VARCHAR(255)    DEFAULT     'Hollywood',
    gender          CHAR(1)         DEFAULT     'U',
    birthdate       DATE            DEFAULT     DATE '1990-08-26'
);
```

```
ALTER TABLE MovieStar ADD phone CHAR(16) DEFAULT 'unlisted';
```

Reminder: Superkeys and Keys

- A **superkey S** for a relation schema R is a subset of the attributes of R such that:
  1. There can't be two different tuples in an instance of R that have the same value for all the attributes in S.

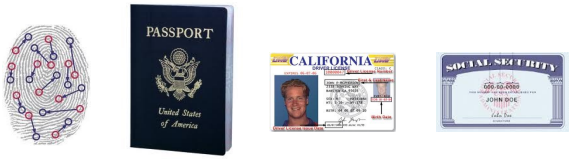
- A **key K** for a relation schema R is a subset of the attributes of R such that:
  1. There can't be two different tuples in an instance of R that have the same value for all the attributes in K.
  2. Minimal: No proper subset of K has the above property.

- All keys are superkeys … but some superkeys are not keys.

- A key is a constraint on the allowable instances of relation R.

Reminder: Key Examples

- Student(studentID, name, major, gender, avgGPA).
  - {studentID} is a key because two different students can't have the same studentId. It is also a superkey.
  - {studentID, name} is a superkey but it's not a key.
  - {studentID, name, major, gender, avgGPA} is a superkey but it's not a key.



- There can be multiple keys in general.
  - One key is chosen and define as the *primary key*, while the rest are *candidate keys*.
- Student(studentID, name, dob, major, gender, avgGPA)
  - {studentID}, {name, dob} are keys and also superkeys.
    - {studentID} is the primary key.
    - {name, dob} is the candidate key.
      - » [Note: This is a questionable toy example.]
    - {name, dob, avgGPA} is a superkey.
  - Can you think of a realistic multi-attribute key for a <u>different</u> table?

Declaring Keys

Two ways to declare a single attribute to be a key in the CREATE TABLE statement:

```
CREATE TABLE MovieStar (
    starName      CHAR(30) PRIMARY KEY,
    address       VARCHAR(255),
    gender        CHAR(1),
    birthdate     DATE
);
```

```
CREATE TABLE MovieStar (
    starname      CHAR(30),
    address       VARCHAR(255),
    gender        CHAR(1),
    birthdate     DATE,
    PRIMARY KEY (starName)
);
```

Declaring Keys (continued)

If the key consists of multiple attributes, then those attributes can be declared as a key as follows:

```
CREATE TABLE Movies (
    movieTitle      CHAR(100),
    movieYear       INT,
    length          INT,
    genre           BOOLEAN,
    studioName      CHAR(30),
    producerC#      INT,
    PRIMARY KEY (movieTitle, movieYear)
);
```

Notation (not SQL): We'll often write
Movies(movieTitle, movieYear, length, genre, studioName, producerC#)
as an informal abbreviation, underlining the Primary Key.

PRIMARY KEY vs. UNIQUE

- In the previous examples, the keyword “PRIMARY KEY” can be replaced by “UNIQUE”.
  - Both specify that attributes are keys, but PRIMARY KEY is not identical to UNIQUE.

```
CREATE TABLE MovieStar (
    starName    CHAR(30),
    address     VARCHAR(255),
    gender      CHAR(1),
    birthdate   DATE,
    PRIMARY KEY (starName)
);
```

```
CREATE TABLE MovieStar (
    starName    CHAR(30),
    address     VARCHAR(255),
    gender      CHAR(1),
    birthdate   DATE,
    UNIQUE (starName)
);
```

- In the standard, SQL Tables aren't required to have a Primary Key, but many implementations do require it (or create it).

PRIMARY KEY vs. UNIQUE (continued)

```
CREATE TABLE MovieStar (
    starName    CHAR(30),
    address     VARCHAR(255),
    gender      CHAR(1),
    birthdate   DATE,
    PRIMARY KEY (starName)
);
```

- None of the rows in MovieStar can have NULL *starName* values.
- Rows are uniquely identified by their *starName* values.
- There can be at most one Primary Key for a table.

```
CREATE TABLE MovieStar (
    starName    CHAR(30),
    address     VARCHAR(255),
    gender      CHAR(1),
    birthdate   DATE,
    UNIQUE (starName)
);
```

- Rows in MovieStar can contain NULL *starName* values.
- Rows in MovieStar with non-NULL *starName* values are uniquely identified by their *name* values.
- There can be multiple unique constraints for a table, in addition to a primary key.

More on NULL

```
CREATE TABLE MovieStar (
    starName        CHAR(30)   PRIMARY KEY,
    address         VARCHAR(255)   NOT NULL  DEFAULT 'Hollywood',
    gender          CHAR(1),
    birthdate       DATE  NOT NULL
);
```

- If no value is entered for an attribute, and no DEFAULT value is specified for that attribute, and that attribute can be NULL, then the value of the attribute will be NULL.
  - NULL is a special value in SQL used to represent unknown values.
  - **NOT NULL** constraint prevents a column from having NULL values.
  - Attributes that don't have NOT NULL specified may be NULL.
  - … but remember that attributes in the PRIMARY KEY can't be NULL.

SQL DDL and SQL DML

- Two main aspects to SQL:
  - Data Definition Language (DDL)
    - Sublanguage of SQL used to create, delete, modify the definition of tables and views.
    - For declaring database schemas.
  - Data Manipulation Language (DML)
    - Sublanguage of SQL that allows users to insert, delete, and modify rows of tables and pose queries to retrieve data.
    - For asking questions about the database and modifying the database.

Reference:
A First Course in Database Systems, 3rd edition, Chapter 6.1

Database Schema for Our Running Example

- Let's assume we have a database schema with five relation schemas.

  Movies(movieTitle, movieYear, length, genre, studioName, producerC#)
  StarsIn(movieTitle, movieYear, starName)
  MovieStar(starName, address, gender, birthdate)
  MovieExec(execName, address, cert#, netWorth)
  Studio(studioName, address, presC#)

In this schema, ***cert#*** *is just an attribute for a MovieExec's "certificate number";* producerC# *and presC# should refer to the cert# of the producer of a Movie and president of a Studio, respectively. Nothing special about these column names/attributes for SQL—it's just an example.*

And note that the ***#*** *symbol is* ***not*** *allowed in identifiers (such as column names and table names) in PostgreSQL, even though our textbook uses it.*

Database Schema: Textbook Differences

- Let's assume we have a database schema with five relation schemas.

  Movies(movieTitle, movieYear, length, genre, studioName, producerC#)
  StarsIn(movieTitle, movieYear, starName)
  MovieStar(starName, address, gender, birthdate)
  MovieExec(execName, address, cert#, netWorth)
  Studio(studioName, address, presC#)

In textbook, schema is a little different:

  Movies(title, year, length, genre, studioName, producerC#)
  StarsIn(movieTitle, movieYear, starName)
  MovieStar(name, address, gender, birthdate)
  MovieExec(name, address, cert#, netWorth)
  Studio(name, address, presC#)

CREATE TABLE for MovieExec

```
CREATE TABLE MovieExec (
    execName        CHAR(30),
    address         VARCHAR(255),
    cert#           INT PRIMARY KEY,
    netWorth        INTEGER
);
```

MovieExec(execName, address, cert#, netWorth)

CREATE TABLE for Studio

```
CREATE TABLE Studio (
    studioName      CHAR(30) PRIMARY KEY,
    address         VARCHAR(255),
    presC#          INT,
    FOREIGN KEY (presC#) REFERENCES MovieExec(cert#)
);
```

Studio(studioName, address, presC#)

- Every presC# in Studio must be the Primary Key (cert#) of a movie exec in the MovieExec table.
  - But for each movie exec in the MovieExec table, there can be n studios in the Studios table for which that movie exec is president, where n can be 0, 1, 2, …, 17, …

CREATE TABLE for Movies

```
CREATE TABLE Movies (
      movieTitle      CHAR(100),
      movieYear       INT,
      length          INT,
      genre           BOOLEAN,
      studioName      CHAR(30),
      producerC#      INT,
      PRIMARY KEY (movieTitle, movieYear),
      FOREIGN KEY (studioName) REFERENCES Studio,
      FOREIGN KEY (producerC#) REFERENCES MovieExec(cert#)
);
```

Didn't have to say:
FOREIGN KEY (studioName)
REFERENCES Studio (studioName),
because referencing attributes and referenced attributes are the same.

Movies(<u>movieTitle, movieYear</u>, length, genre, studioName, producerC#)

- Every studioName in Movies must be the Primary Key (studioName) of a studio in the Studio table.
  - But for each studioName in the Studio table, there can be n movies in the Movies table which have that studioName, where n can be 0, 1, 2, …, 17, …
- Every producerC# in Movies must be the Primary Key (cert#) of a movie exec in the MovieExec table.
  - But for each movie exec in the MovieExec table, there can be n movies in the Movies table for which that movie exec is producer, where n can be 0, 1, 2, …, 17, …

CREATE TABLE for MovieStar

```
CREATE TABLE MovieStar (
    starName        CHAR(30) PRIMARY KEY,
    address         VARCHAR(255),
    gender          CHAR(1),
    birthdate       DATE
);
```

MovieStar(<u>starName</u>, address, gender, birthdate)

CREATE TABLE for StarsIn

```
CREATE TABLE StarsIn (
    movieTitle CHAR(100),
    movieYear INT,
    starName CHAR(30),
    PRIMARY KEY (movieTitle, movieYear, starName),
    FOREIGN KEY (movieTitle, movieYear) REFERENCES Movies,
    FOREIGN KEY (starName) REFERENCES MovieStar
);
```

StarsIn(<u>movieTitle, movieYear, starName</u>)

- Every (movieTitle, movieYear) in StarsIn must be the Primary Key of a movie in the Movies table.
  - But for each (movieTitle, movieYear) in the Movies table, there can be n Movies in the StarsIn table which have that (movieTitle, movieYear) , where n can be 0, 1, 2, …, 17, …
- Every starName in StarsIn must be the Primary Key of a movie star in the MovieStars table.
  - But for each starName in the MovieStar table, there can be n Movies in the StarsIn table which have that starName, where n can be 0, 1, 2, …, 17, …

PostgreSQL Meta Commands

- \l
  - List the databases.

After you connect to a database, you can perform the following:

- \dt
  - List the tables of a database.
- \d table_name
  - List the columns for a specific table_name.

Tables are not necessarily listed in the order in which you created them. That's okay!

Identifiers, such a table names and column names will appear in lowercase, not in the case you used when you created them. That's okay!

You can use the “double quote” symbol (“) around an identifier if you really care about case … but then you'll always have to use double quotes around that identifier.

Example: “movieYear” versus movieYear

https://www.geeksforgeeks.org/postgresql-psql-commands/

A Simple SQL Query

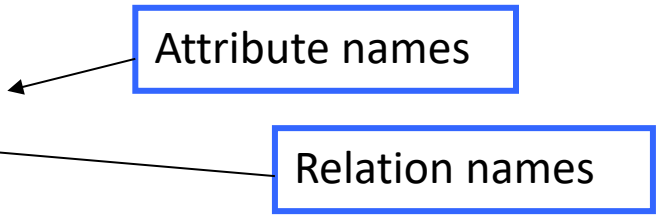
- Find all movies produced by Disney Studios in 1990.

```
SELECT *
FROM  Movies
WHERE  studioName = 'Disney' AND movieYear = 1990;
```

Basic Form of a SQL Query

- Basic form:

SELECT [DISTINCT] $c_1, c_2, \ldots, c_m$
FROM $R_1, R_2, \ldots, R_n$
[WHERE *condition*]



- We will focus on one relation R_1 for now.
- What is the semantics (that is, the meaning) of this query?

A Simple SQL Query

- Find all movies produced by Disney Studios in 1990.

```
SELECT *
FROM Movies
WHERE studioName = 'Disney' AND movieYear = 1990;
```

The symbol "*" is a shorthand for all attributes of relations in the FROM clause.

Movies

movieTitle	movieYear	length	genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999
Monster's Inc.	1990	121	true	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

A Simple SQL Query (Result)

- Find all movies produced by Disney Studios in 1990.

```
SELECT *
FROM Movies
WHERE studioName = 'Disney' AND movieYear = 1990;
```

Result

movieTitle	movieYear	length	genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999

An Even Simpler SQL Query

- Find all movies.

  SELECT *

  FROM Movies;

Equivalent to:

```
SELECT *
FROM Movies
WHERE TRUE;
```

Movies

movieTitle	movieYear	length	genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999
Monster's Inc.	1990	121	true	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

An Even Simpler SQL Query (Result)

- Find all movies.

```
SELECT *
FROM Movies;
```

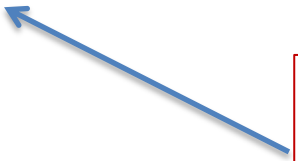
Result

movieTitle	movieYear	length	genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999
Monster's Inc.	1990	121	true	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

A Simple SQL Query with Projection

- Return the title and year of all movies.

```
SELECT movieTitle, movieYear
FROM Movies;
```



A Projection Query:
- Only a subset of attributes from the relation(s) in the FROM clause is selected.

Result

movieTitle	movieYear
Pretty Woman	1990
Monster's Inc.	1990
Jurassic Park	1998

A Simple SQL Query with Projection and Selection

- Return the title and year of all movies produced by Disney Studios in 1990.

```
SELECT movieTitle, movieYear
FROM Movies
WHERE studioName = 'Disney' AND movieYear = 1990;
```

Result

movieTitle	movieYear
Pretty Woman	1990

Distinct: Sets vs Multisets/Bags

- Return the years of all movies with length less than 140.

```
SELECT movieYear
FROM Movies
WHERE length < 140;
```

```
SELECT DISTINCT movieYear
FROM Movies
WHERE length < 140;
```

Result

movieYear
1990
1990

Multiset or bag semantics

Result

movieYear
1990

Set semantics

Bags (or Multisets) vs. Sets

- From basic set theory:
- Every element in a set is distinct
  - E.g., {2,4,6} is a set but {2,4,6,2,2} is not a set
    - … or is same set as {2,4,6}
- A bag (or multiset) may contain repeated elements.
  - E.g., {{2,4,6}} is a bag. So is {{2,4,6,2,2}}.
    - Note that double set brackets in {{2,4,6}} indicate it's a bag, not a set
  - Equivalently written as {{2[3],4[1],6[1]}}.
- The order among elements in a set or bag is not important
  - E.g., {2,4,6} = {4,2,6} = {6,4,2}
  - {{2,4,6,2,2}} = {{2,2,2,6,4}} = {{6,2,2,4,2}}.

Aliasing Attributes

- Allows you to rename the attributes of the result.
- Example: Return the title and length of all movies as attributes name and duration.

```
SELECT movieTitle AS name, length AS duration
FROM Movies;
```

The keyword AS is optional when you alias in SELECT clause.

Result

name	duration
Pretty Woman	119
Monster's Inc.	121
Jurassic Park	145

Expressions in the SELECT Clause

- Expressions are allowed in the SELECT clause.
- Return the title and length of all movies as name and duration in seconds (durationInSeconds).

```
SELECT movieTitle AS name,
        length * 60 AS durationInSeconds
FROM Movies;
```

Result

name	durationInSeconds
Pretty Woman	7140
Monster's Inc.	7260
Jurassic Park	8700

Constants in the Result

- Constants can also be in the SELECT clause.
- Every row will have the same constant specified in the SELECT clause.

```
SELECT movieTitle AS name, length * 60  AS
    durationInSeconds, 'seconds' AS inSeconds
FROM Movies;
```

Result

name	durationInSeconds	inSeconds
Pretty Woman	7140	seconds
Monster's Inc.	7260	seconds
Jurassic Park	8700	seconds

More on Conditions in the WHERE Clause

WHERE studioName = 'Disney' AND movieYear = 1990;

- Comparison operators:
  - =, <>, <, >, <=, >=, LIKE
  - Equal, not equal, less than, greater than, less than or equal, greater than or equal, like
  - Example:
    WHERE movieYear <= 1990
- Logical connectives:
  - AND, OR, NOT
  - Examples:
    WHERE movieYear >= 1990 AND movieYear <= 2005
    That can also be written as: WHERE movieYear BETWEEN 1990 AND 2005
    WHERE NOT (studioName = 'DISNEY' AND movieYear <= 1990)
- Arithmetic expressions:
  - +, -, *, /, etc.
  - Example:
    WHERE ((length/60) > 2 OR (length < 100)) AND movieYear > 2000

Can use != instead of <>

More on the Conditions in the WHERE Clause (cont'd)

- In general, the WHERE clause consists of a boolean combination of conditions using logical connectives AND, OR, and NOT.
- Each condition is of the form

  expression op expression

  - *expression* is a column name, a constant, or an arithmetic or string expression
  - op is a comparison operator (=, <>, <, >, <=, >=, LIKE)

(More on this very soon)

String Comparisons

- Strings are compared in lexicographical order.
  - Let $a_1.a_2. \ldots .a_n$ and $b_1.b_2. \ldots b_m$ be two strings.
  - Then $a_1.a_2. \ldots .a_n < b_1.b_2. \ldots b_m$ if either:
    1. $a_1.a_2. \ldots .a_n$ is a proper prefix of $b_1.b_2. \ldots b_m$, or,
    2. For some $1 \leq i \leq n$, we have $a_1=b_1$, $a_2=b_2$, …, $a_{i-1}=b_{i-1}$ and $a_i < b_i$.

- Examples:
  - 'Pretty' < 'Pretty Woman',
  - 'butterfingers' < 'butterfly'

Pattern Matching in SQL

- LIKE operator
  - s LIKE p, s NOT LIKE p
  - s is a string, p is a pattern

- '%' (stands for 0 or more arbitrary chars)
- '_' (stands for exactly one arbitrary char)

```
SELECT *
FROM Movies
WHERE movieTitle LIKE 'Jurassic%'
```

```
SELECT *
FROM Movies
WHERE movieTitle LIKE 'Juras_ic%'
```

Pattern Matching in SQL

- How do you match titles with a quote symbol in them?
  - E.g., 'Monster's Inc.'

```
SELECT *
FROM Movies
WHERE movieTitle LIKE 'Monster's Inc.'
```

Wrong!!!

```
WHERE movieTitle LIKE 'Monster''s Inc.'
```

Pattern Matching in SQL

- How do you match just a quote symbol?

  WHERE movieTitle LIKE '''' // matches ' (single quote)

- How do you match two quote symbols?

  WHERE movieTitle LIKE '''''' // matches '' (two single quotes)

- How do you match % or _?

  WHERE movieTitle LIKE '!%%!_' ESCAPE '!'

  // ! could be any character

Would this match: 'ABC%D' '%ABC' '%ABC_' 'ABCD' '%_'

No No Yes No Yes

- You can even match quote using an ESCAPE character.

  WHERE movieTitle LIKE ':''' ESCAPE ':' // matches '' (two single quotes)

Comparing Dates and Times

- Reminder about DATE, TIME, TIMESTAMP and INTERVAL
  - Separate data types
  - Constants are character strings of a specific form
    - DATE '2021-01-13' or DATE '01/13/21'
    - TIME '16:45:33'
    - TIMESTAMP '2021-01-13 16:45:33'
    - INTERVAL '2 HOURS 30 MINUTES'
  - See Chapter 6.1.5 of Textbook or PostgreSQL manual
- DATE, TIME, TIMESTAMP and INTERVAL types can be compared using ordinary comparison operators.
  - … WHERE ReleaseDate <= DATE '2020-06-19' ...
- There are some implementation-specific differences on formats.
- Localization of types, e.g., for European dates
  - Character sets can also be localized.

SQL's Three-Valued Logic

Students

studentID	name	major	gender	avgGPA
112	Ann	Computer Science	F	NULL
327	Bob	NULL	M	3.90
835	Carl	Physics	M	4.00

- What is the result of this comparison?

  major = 'Computer Science' AND avgGPA > 3.00

- Three-valued logic TRUE, FALSE, UNKNOWN
- If major is NULL, then " major= 'Computer Science' " evaluates to UNKNOWN.
- If avgGPA is NULL, then " avgGPA > 3.00 " evaluates to UNKNOWN.

SQL's Three-Valued Logic (cont'd)

p	*q*	*p* OR *q*	*p* AND *q*	*p* = *q*
True	True			
True	False			
True	Unknown			
False	True			
False	False			
False	Unknown			
Unknown	True			
Unknown	False			
Unknown	Unknown			

p	NOT *p*
True	
False	
Unknown	

SQL's Three-Valued Logic: Truth Table

p	*q*	*p* OR *q*	*p* AND *q*	*p* = *q*
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

p	NOT *p*
True	False
False	True
Unknown	Unknown

Example of Three-Value Logic

```
SELECT *
FROM Students
WHERE major = 'Computer Science' AND avgGPA > 3.00;
```

- If the condition evaluates to UNKNOWN, then the tuple will not be returned.
- Both Ann and Bob will not be returned.
- In fact, none of the tuples will be returned.

- Is UNKNOWN the same as FALSE?
  - No; why not?
  - What happens if the condition was the following?

    NOT (major = 'Computer Science' AND avgGPA > 3.00)

Some Facts About NULL

- Almost all comparisons with NULL will evaluate to UNKNOWN. If salary is NULL, then the following will be UNKNOWN:
  - salary = 10
  - salary <> 10
  - 90 > salary OR 90 <= salary
  - salary = NULL
  - salary <> NULL
- Use of IS NULL and IS NOT NULL
  - salary IS NULL
    will be TRUE if salary is NULL, FALSE otherwise
  - salary IS NOT NULL
    will be TRUE if salary isn't NULL, FALSE otherwise

Some More Examples For NULL

- If salary1 has value NULL and salary2 has value NULL, then what will be the value for these predicates?
  - salary1 = salary2
  - salary1 <> salary2
  - salary1 IS NULL
  - salary2 IS NOT NULL
  - salary1 IS NULL OR salary2 IS NOT NULL
- Write a query that returns the names of students whose Major is either Computer Science or NULL.
- Write a query that returns the names of students whose Major isn't NULL and whose avgGPA is NULL.

SQL's Three-Valued Logic

Students

studentID	name	major	gender	avgGPA
112	Ann	Computer Science	F	NULL
327	Bob	NULL	M	3.90
835	Carl	Physics	M	4.00

- What is the result of this comparison?
  - major = 'Computer Science' AND avgGPA > 3.00

- Three-valued logic TRUE, FALSE, UNKNOWN
- If major is NULL, then " major= 'Computer Science' " evaluates to UNKNOWN.
- If avgGPA is NULL, then " avgGPA > 3.00 " evaluates to UNKNOWN.

NULL, UNKNOWN and Empty Set

These are very different!

- An **Attribute** can have value **NULL**
- The result of a **Condition** can be **UNKNOWN**
  - SQL's Three-Valued Logic:
    Not just TRUE and FALSE, but also UNKNOWN.
- The **Result of a Query** can be the **Empty Set**, meaning that there are no tuples in the result.
  - A table may also have no tuples in it.

Ordering the Result

SELECT [DISTINCT] <list of attributes>
FROM $R_1, R_2, \ldots, R_n$
[WHERE *condition*]
[ORDER BY <list of attributes>]

- ORDER BY presents the result in a sorted order.
- By default, the result will be ordered in ascending order, ASC.
  - For descending order on an attribute, you write DESC next to that attribute in the list of attributes.
  - Example appears on next slide.

ORDER BY

```
SELECT *
FROM Movies
WHERE studioName = 'Disney' AND movieYear = 1990
ORDER BY length, movieTitle;
```

- The result will list movies that satisfy the condition in the WHERE clause in ascending (ASC) order of length, then by ascending title.
  - Shortest length movies will be listed first.
  - Among all movies with the same length, the movies will be sorted in ascending order of movieTitle.
- ORDER BY length DESC, movieTitle;
  - Longest length movies will be listed first.
  - Among all movies with the same length, the movies will be sorted in ascending order of movieTitle.
- What about: ORDER BY length, movieTitle DESC;

More on ORDER BY

- You can ORDER BY expressions, not just attributes
  - ORDER BY Quantity * Price
  - ORDER BY Quantity * Price DESC

- ORDER BY works with attributes that can have NULL values
  - NULL will probably be smallest or largest value.
  - Not specified by SQL standard, so it depends on the implementation.
    - For example, in PostgreSQL, DB2 and Oracle, NULL is the largest value, but in SQL Server and SAP HANA, NULL is the smallest value.
  - For purposes of questions on ORDER BY in a Gradiance assignment, please treat NULL as the smallest possible value, not the largest possible value.
    - If an exam requires ORDER BY with NULL, you'll be told whether NULL should be treated as the smallest or the largest value.

Meaning of an SQL Query with One Relation in the FROM Clause

SELECT [DISTINCT] $c_1, c_2, \ldots, c_m$
FROM R_1
[WHERE *condition*]
[ORDER BY < list of attributes [ASC | DESC] >]

- Let Result denote an empty multiset of tuples.
- For every tuple *t* from R_1,
  - if *t* satisfies *condition (*i.e., if condition evaluates to true), then add the tuple that consists of $c_1, c_2, \ldots, c_m$ components of *t* into Result.
- If DISTINCT is in the SELECT clause, then remove duplicates in Result.
- If ORDER BY <list of attributes> exists, then order the tuples in Result according to ORDER BY clause.
- Return Result.

Practice Homework 2

- Define the following two relational schemas in SQL using reasonable datatypes for each attribute.
  - Sells (bar, beer, price): indicates the price of each beer sold at each bar. (Note that each bar can sell many beers and many bars can sell the same beer, at possibly different prices.)
  - Frequents (drinker, bar): indicates which drinkers frequent which bars. (Note that each drinker may frequent many bars and many drinkers may frequent the same bar.)
  - Likes (drinker, beer): indicates which drinker likes which beers. (Note that a drinker may like many beers, and many drinkers may like the same beer.)
- Add a column "size" to Sells relation schema using the "ALTER TABLE" command.
- Write an SQL query to retrieve all beers sold by the bar "99 bottles".
- Write an SQL query to retrieve all beers that are priced above $3.