Basic SQL

Instructor: Shel Finkelstein

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

Important Notices

- You should get Gradiance access this week, via Labs with TAs.
  - First Gradiance Assignment will be posted soon
- Lab1 assignment was posted Friday, Jan 13 on Piazza under Resources, together with General Instructions.   See Piazza announcement. Read it soon!
  - You should already have PostgreSQL access.
  - Lab1 will be discussed at next week's Labs.
  - Due by **Sunday, Jan 22**, by 11:59pm on Canvas
    - Be sure to understand instructions for Lab1, including use of PosgreSQL, ftp, zip and Canvas.
    - <u>No</u> late submissions, <u>no</u> make-up assignments
  - Canvas requires submission of zip file for Lab1.
    - Canvas will also be used for grading of Labs and Exams.

Accessing Canvas

LOGIN INFORMATION FOR CANVAS FALL COURSES

1. Go to http://login.uconline.edu/

2. Click on UC Campus Student and Faculty > UC Santa Cruz.

3. Log in with your username and Gold password.

4. You will see your Dashboard. Click on the course you want.

Please ensure that you have access. Lab1 was posted to Piazza, but Lab1 grading will be on Canvas.

Practice Homework

- If D_1 has n_1 elements and D_2 has n_2 elements, then how many elements are there in $D_1 \times D_2$?

    - That is, if $|D_1| = n_1$, $|D_2| = n_2$, what is $|D_1 \times D_2|$?

- 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_i has n_i elements, then how many relations can one construct from $D_1 \times \ldots \times D_k$?

Summary of Previous Lecture

- A data model
- A relation schema
  - Attributes or column names
  - Tuples or rows
  - Columns
  - Arity (number of columns/attributes)
- A relation (an instance of a relation schema)
- A relational database schema
- A database (an instance of a database schema)
- Logical and physical data independence

A Relation Database Schema

- A *relation 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, ... R_k} (or a *database instance* in short) is a set {r_1, ..., 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	Algo.	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

- Three 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 constructed only when needed.
    3. Temporary tables
        - These are (intermediate) tables that are constructed by the SQL execution engine during the processing of SQL queries and discarded when done.

Most of the Primitive Data Types in SQL

- CHAR(n): fixed-length string of up to n characters (blank-padded)
- VARCHAR(n): also a string of up to n characters
- BIT(n)
- BIT VARYING(n)
- BOOLEAN: true/false   (unknown)
- INT or INTEGER
  - Analogous to int in C
- SHORTINT
  - Analogous to short int in C

Primitive Data Types in SQL (continued)

- DECIMAL(n,d)
  - 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 and TIME (and TIMESTAMP)
  - Separate data types
  - Constants are character strings of a specific form, e.g., 2015-01-13 and 16:45:33 for DATE and TIME
- A few others …
- PostgreSQL has non-standard TEXT, for variable strings of any length

Defining a Table

```
CREATE TABLE Movies (
    title           CHAR(100),
    year            INT,
    length          INT,
    genre           CHAR(10),
    studioName      CHAR(30),
    producerC#      INT
);
```

| title	year	length	genre	studioName	producerC#

Defining a Table (continued)

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

name	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
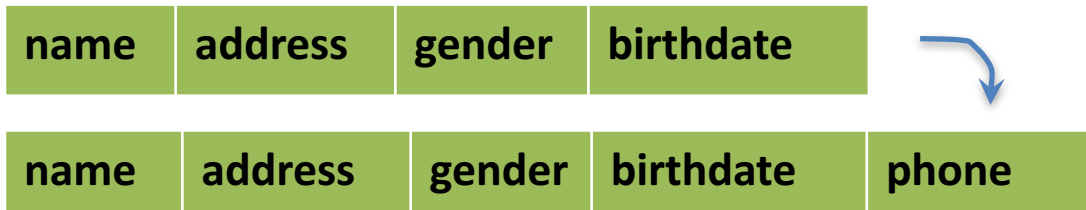
- ALTER TABLE MovieStar DROP birthdate;

name	address	gender	phone

Default Values

CREATE TABLE MovieStar (

    name   CHAR(30),

    address   VARCHAR(255)   **DEFAULT** 'Hollywood',

    gender   CHAR(1),

    birthdate   DATE

);

- If a new row is inserted and no value is specified for the attribute address, then the value for this attribute will default to 'Hollywood'.

- If no default value is declared explicitly and no value is entered for an attribute, then the value of the attribute will default to NULL.

  - NULL is a special value in SQL to represent unknown values.

  - A Constraint (which we'll discuss soon) can prevent a column from having null values.

More Examples of Default Values

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



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

Reminder: Keys

- A *key constraint* (or a *key* in short) of a relation schema R is a subset K of attributes of R such that

  1. For every instance r of R, every two distinct tuples of r must differ in their values from K.

     - Contrapositive: if two tuples agree on their values from K, then they must be the same tuple.

  2. Minimal: no proper subset of K has the above property.

- A *superkey* is a set of attributes of R that includes a key of R.

  – <u>Fact</u>. All keys are superkeys but some superkeys are not keys.

Reminder: Key Examples

– Student(studentID, name, major, gender, avgGPA).

- {studentID} is a key. It is also a superkey.
- {studentID, name} is a superkey but not a key.
- {studentID, name, major, gender, avgGPA} is a superkey but 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.
- {name, dob, avgGPA} is a superkey.

Declaring Keys

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

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

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

Declaring Keys (continued)

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

```
CREATE TABLE Movies (
    title        CHAR(100),
    year         INT,
    length       INT,
    genre        CHAR(10),
    studioName   CHAR(30),
    producerC#   INT,
    PRIMARY KEY (title, year)
);
```

PRIMARY KEY vs. UNIQUE

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

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

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

- In the standard, SQL Tables aren't required to have a key (primary or otherwise), but some implementations require it (or create it).

PRIMARY KEY vs. UNIQUE (continued)

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

- None of the rows in MovieStar can have null *name* values.

- Rows are uniquely identified by their *name* values.

- There can be at most one primary key for a table.

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

- Rows in MovieStar <u>can</u> contain null *name* values.

- Rows in MovieStar with <u>non-null</u> *name* 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 (

    name   CHAR(30) **PRIMARY KEY**,

    address   VARCHAR(255) **NOT NULL** **DEFAULT** 'Hollywood',

    gender   CHAR(1),

    birthdate   DATE **NOT NULL**

);

- If no default value is declared explicitly and no value is entered for an attribute, then the value of the attribute will default to NULL.
  - NULL is a special value in SQL 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.

PostgreSQL Meta Commands

- \l
  - List the databases.

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

- \d
  - List the tables of a database.

- \d table
  - List the columns of a table.

http://www.postgresqlforbeginners.com/2010/11/interacting-with-postgresql-psql.html

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(title, year, length, genre, studioName, producerC#)
    StarsIn(movieTitle, movieYear, starName)
    MovieStar(name, address, gender, birthdate)
    MovieExec(name, address, cert#, netWorth)
    Studio(name, 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 attribute names for SQL—it's just an example.*

A Simple SQL Query

- Find all movies produced by Disney Studios in 1990.

  SELECT *

  FROM Movies

  WHERE studioName = 'Disney' AND year = 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*]

  Attribute names

  Relation names

- 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 year = 1990;

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

Movies

Title	Year	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 year = 1990;

Result

Title	Year	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

Title	Year	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

Title	Year	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 title, year

    FROM Movies;

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

Result

Title	Year
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 title, year

  FROM Movies

  WHERE studioName = 'Disney' AND year = 1990;

Result

Title	Year
Pretty Woman	1990

Distinct: Sets vs Multisets/Bags

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

SELECT year
FROM Movies
WHERE length < 140;

SELECT **DISTINCT** year
FROM Movies
WHERE length < 140;

Result

Year
1990
1990

Result

Year
1990

Multiset or bag semantics

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 title AS name, length AS duration

  FROM Movies;

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 title 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 included in the SELECT clause.
- Every row will have the same constant specified in the SELECT clause.

SELECT title 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 the Conditions in the WHERE Clause

- WHERE studioName = 'Disney' AND year = 1990;

- Comparison operators:
    - =, <>, <, >, <=, >=
    - Equal, not equal, less than, greater than, less than or equal, greater than or equal
    - E.g., WHERE year <= 1990

- Logical connectives:
    - AND, OR, NOT
    - E.g., WHERE NOT (studioName = 'DISNEY' AND year <= 1990)

- Arithmetic expressions:
    - +, -, *, /, etc.
    - E.g., WHERE ((length*0.01667) > 2 OR (length < 100)) AND year > 2000

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 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 $1 <= i < n$, 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 title LIKE 'Pretty%'   title LIKE 'P_etty%'

Pattern Matching in SQL

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

SELECT *

FROM Movies

WHERE title LIKE 'Monster's Inc.'

Wrong!!!

WHERE title LIKE 'Monster''s Inc.'

Pattern Matching in SQL

- How do you match just a quote symbol?

WHERE title LIKE ''''   // matches '

- How do you match two quote symbols?

WHERE title LIKE ''''''   // matches ''

- How do you match % or _?

WHERE title LIKE '!%%!_' ESCAPE '!' // ! could be any character

Would this match: 'ABC%D'   '%ABC'   '%ABC_'   'ABCD'   '%_'
                              No   No   Yes   No   Yes

Date and Time

- DATE and TIME (and TIMESTAMP)
  - Separate data types
  - Constants are character strings of a specific form
    - DATE '2015-01-13'
    - TIME '16:45:33'
    - TIMESTAMP '2015-01-13 16:45:33'
  - DATE, TIME and TIMESTAMP can be compared using ordinary comparison operators
    - … WHERE ReleaseDate <= DATE '1990-06-19' …
  - See Chapter 6.1.5 of Textbook (page 251) for some more details.
  - There are some implementation-specific differences on formats.

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

- Three-valued logic TRUE, FALSE, UNKNOWN
- If major is NULL, then " major= 'Computer Science' " evaluates to UNKNOWN.
- If avgGPA is NULL, then "avgGPA > 3" 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 > 3.0;

- If the condition evaluates to UNKNOWN, then the tuple will <u>not</u> be returned.

- Both Ann and Bob will <u>not </u>be returned.

- In fact, none of the tuples will be returned.

- Is UNKNOWN the same as FALSE?
  - No; why not?

Some Facts About Nulls

- 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 NOT IS 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 Nulls

- If Salary1 has value NULL and Salary2 has value NULL, then what will be the value for these predicates?
  - Sarlary1 = 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

- Three-valued logic TRUE, FALSE, UNKNOWN
- If major is NULL, then " major= 'Computer Science' " evaluates to UNKNOWN.
- If avgGPA is NULL, then "avgGPA > 3" evaluates to UNKNOWN.

Ordering the Result

SELECT [DISTINCT] <list of attributes>
FROM $R_1, R_2, …, R_n$
[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. For descending order, you write:
    – ORDER BY <list of attributes> DESC

ORDER BY

SELECT *

FROM Movies

WHERE studioName = 'Disney' AND year = 1990

ORDER BY length, title;

- 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 title.
- ORDER BY length DESC, title;
  - <u>Longest</u> length movies will be listed first.
  - Among all movies with the same length, the movies will be sorted in ascending order of title.

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

SELECT [DISTINCT] c_1, c_2, …, 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, …, 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.

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

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 drinker frequents 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.