DEPAUL UNIVERSITY
COLLEGE OF COMPUTING AND DIGITAL MEDIA

CSC 202 NOTES
Spring 2010: Amber Settle

Week 2: Monday, April 5, 2010

Announcements

- First assignment is due now
- The second assignment is due Monday, April 12th
- Questions about the assignment?
  - The graded assignments will be handed back/comments posted by next week
  - I will go over the most commonly missed questions next week

More database logic

A database typically consists of more than one table.

The full reason for this is beyond the scope of the course, but a short answer is that a single table database with any reasonable amount of information would have a great deal of duplication.

For example, our (very small) student database has information about students, courses, and which courses the students are enrolled in.

To use a particular example, this single table database would have to have **two rows for Abigail Winters**, since in our database we have two records of courses for her.

In a single table, all of her personal information would be duplicated for each of those records:

LN	..	Cr	City	Start	CID	…	Dpt	Nr	…
Winters	..	GRD	Chicago	2003	1092	…	CSC	440	…
Winters	..	GRD	Chicago	2003	1020	…	CSC	489	…

The problem is then with **consistency** (and efficiency)!

For example, suppose Abigail Winters moves. Then we need to update her city, and we need to do so in all the rows for each class she's taken.

That will mean 13+ updates for graduate students and more for undergraduates. Also, if we do the updates incorrectly then we'll

have multiple pieces of information for Abigail Winters, which can contaminate our database.

The sample database consists of 5 tables – see the **StudentDatabase.pdf** file posted in the Documents on COL.

Look at the Enrolled table. What does the second row in that table mean?

StudentID	CourseID	Quarter	Year
11035	1020	Fall	2005

The first entry is the student's ID and the second is the course ID.

So this first row means that student with ID 11035 is enrolled in the course with ID 1092 in Fall 2005.

Looking up the student and course information, this means that Abigail Winters is enrolled for cryptography in the Fall 2005 quarter.

In database terminology, StudentID and CourseID in the Enrolled table are **foreign keys**, pointing at the primary key fields of another table and identifying a particular record in that table.

In this case, StudentID points to the primary key SID in the Student table, and CourseID points to the primary key CID in the Courses table.

Exercises:

1. Interpret another row in the Enrolled table.
2. What would you have to add to the database to record that Peter Johnson took the database course IT 240 in the Spring 2004 quarter?

Combining tables

In SQL you can **choose multiple tables** in the FROM clause of the SELECT statement.

Doing this will create all possible combinations of records from the tables. For this reason we need to limit the output to meaningful records.

For example, if we wrote:

```
SELECT LastName, SID, CID, CourseName
FROM Student, Enrolled, Course
```

We would get 8 * 6 * 7 = 336 records, including many that don't make any sense.

We only want records where the **ID information between the Enrolled table and the other two tables matches**.

To do this we write:

```
SELECT LastName, SID, CID, CourseName
FROM Student, Enrolled, Course
WHERE SID = StudentID AND CourseID = CID;
```

This also explains why we gave the student ID two different names in the Student and Enrolled tables.

If we hadn't we would have to write: Student.SID = Enrolled.SID in place of what we have above which is cumbersome.

New task: Write a query to list all students who are enrolled in the theory of computation course.

We need to combine the Student, Enrolled, and Course tables.

But we need to add a restriction that the students are in theory of computation:

```
SELECT LastName, FirstName, SID
FROM Student, Enrolled, Course
WHERE SID = StudentID AND CourseID = CID AND
    CourseName = 'Theory of Computation';
```

How many courses with the title Theory of Computation are there in the database? What if we only wanted CSC 489?

```
SELECT LastName, FirstName, SID
FROM Student, Enrolled, Course
WHERE SID = StudentID AND CourseID = CID AND
    Department = 'CSC' AND CourseNr = '489';
```

What is the **general approach to use when writing a database query**?

1. Determine which tables are involved (FROM clause)
2. Determine how to connect the foreign keys in those tables with the attributes they point at (first part of the WHERE clause)
3. Add specific requirements (remainder of the WHERE clause)

Final example: Find all students who have not taken theory of computation.

A first attempt might be:

```
SELECT LastName, FirstName, SID
FROM Student, Enrolled, Course
WHERE SID = StudentID AND CourseID = CID AND
        NOT CourseName = 'Theory of Computation';
```

But this returns Abigail Winters, who has taken Theory of Computation. Why?

We can't yet solve this problem. We'll return to it later.

Sets and elements

A **set** is a collection of elements.

Examples:

- C = {Charlie, Djengo, Hugin, Joon, Marcel, Simone}
- P = {2, 3, 5, 7, 11, …}

To show element **e is an element** of a set S, we write: e ∈ S
To show element **e is not an element** of a set, we write e ∉ S

Examples:

- Djengo ∈ C where C is defined above
- Meg ∉ C
- 8 ∉ P where P is defined above

We write |X| for the **cardinality** of X, that is, the number of elements X contains.

Example:

- |C| = 6

Determining the cardinality of a set means counting the number of records. SQL allows you to do that too.

Counting in SQL

The following examples count various sets found in the sample university database.

```
SELECT count(*)
FROM Student;
```

Counts the number of students.
count(*) means to count the number of records.

```
SELECT count(*)
FROM  Student, Enrolled, Course
WHERE SID = StudentID AND CourseID = CID AND
        Department = 'CSC' AND year = 2005;
```

Gives the total enrollment in CSC courses in the year 2005.
Note that students who were enrolled in multiple CSC classes will be counted more than once.

```
SELECT count(DISTINCT SID)
FROM  Student, Enrolled, Course
WHERE SID = StudentID AND CourseID = CID AND
        Department = 'CSC' AND year = 2005;
```

Counts the number of distinct students enrolled in CSC courses in the year 2005.

More terminology

Two sets are **equal** if they contain the same elements.

Note that the order of elements in sets is arbitrary.
Also, it doesn't matter if we list elements more than once – the set is still the same.

Examples:

- {Djengo, Joon, Simone} = {Simone, Djengo, Joon}
- {Djengo, Djengo, Djengo} = {Djengo}

In general you should write the shortest version of the set.
e.g. {Djengo, Hugin} not {Djengo, Hugin, Djengo}

Two sets can be related by the subset relationship.

A is a **subset** of B, written $A \subseteq B$, if every element of A is also an element of B.

Examples:

- {Djengo, Hugin} ⊆ {Djengo, Hugin, Simone}
- {Djengo, Simone} ⊆ {Charlie, Djengo, Hugin, Joon, Marcel, Simone}
- {Djengo, Hugin} ⊈ {Simone, Hugin} since Djengo is an element of {Djengo, Hugin} but is not an element of {Simone, Hugin}

Exercises: Which of the following are true? Why or why not?

- {Djengo, Hugin} ⊆ {Hugin, Djengo, Simone}
- {Charlie, Djengo, Hugin, Joon, Marcel} ⊆ {Charlie, Hugin, Joon, Marcel}
- {Djengo, Joon} ⊆ {Joon, Djengo}
- {Joon, Joon} ⊆ {Joon}

There is a special set, called **the empty set**, that does not contain any elements. It is written {} or ϕ.

Finally, we need to consider an alternative way of specifying sets.

Until now we have listed sets by listing all the elements of the set between opening and closing brackets. This is what is called the **extensional definition**.

This can be very cumbersome for large sets and impossible for infinite sets.

An alternative approach is to give an **intensional definition** of a set, that is, to specify a set as a collection of all elements with a particular property.

Examples:

- P = {x : x is prime}
- C = {c: c is/was a cat who belonged to Amber Settle}

Venn diagrams

The relationship between two sets can be represented using pictures, called Venn diagrams.

For example, there are **two ways to represent the relationship A ⊆ B for two sets A and B**:

1. A ⊂ B
2. A = B

See the drawings on page 257 of the Epp textbook.

In another example, there are **three ways to represent the relationship A ⊈ B**:

1. A and B have no overlap
2. A and B have some overlap
3. B ⊂ A

Again, see the drawings on page 257 of the Epp textbook.

Basic results about sets

The following basic results about sets are useful and fairly easy to prove.

> **Lemma (Modus Barbara)**: If $A \subseteq B$ and $B \subseteq C$ then $A \subseteq C$.
> **Proof**: We have to show that every element of A is an element of C. Suppose $x \in A$. Then $x \in B$ by assumption since $A \subseteq B$. Then $x \in C$ again by assumption since $B \subseteq C$. That means that every element of A is also an element of C, which is what we were required to prove.
>
> > Note: This proof was essentially just a translation of the definitions. Remember that was the second piece of advice Maass gave us!

The next result talks about the empty set.

> **Lemma**: $\phi \subseteq B$ for every set B.
> **Proof**: By the definition of subset, $A \subseteq B$ is true if every element of A is also contained in B. So we need to make sure that every element of ϕ is also an element of B. But ϕ does not have any elements, this is always true, whatever set B is.

Finally, let's work on a proof ourselves.

> **Exercise**: Show that $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$.
>
> Note: Does this remind you of anything from our work with propositional logic?

Back to SQL

The set theory we have seen so far gives us a way to write better SQL queries.

> Suppose we want all students in computer science, information systems, and computer game development.
>
> > We could extend the construct we saw the first week by adding an OR for the computer game development students.
>
> But **SQL has an equivalent for the set theoretic relation is an element of called IN**.
>
> > We also use parentheses instead of curly braces, but otherwise the way it is written is the same as set theory.

To find **all students in computer science, information systems, and computer game development**:

```
SELECT LastName, FirstName, SID
FROM Student
WHERE Program IN ('COMP-SCI', 'INFO-SYS', 'COMP-GAM');
```

To find **all students who took "Theory of Computation" after 1995** using our logical approach we would write:

```
SELECT LastName, FirstName, Student.SID
FROM Student, Enrolled, Course
WHERE SID = StudentID AND CourseID = CID AND
        CourseName = 'Theory of Computation' AND
        Year > 1995;
```

The problem with this is that we're mixing the presentation (the student name associated with the student ID) and the information needed for the query (the student ID).

We can separate things if we think of **the query in two parts**:

1. Get the student IDs of all students taking "Theory of Computation" after 1995
2. Display the names and student IDs from the set found in step 1

Part 1 is:

```
SELECT StudentID
FROM Enrolled, Course
WHERE CourseID = CID AND
        CourseName = 'Theory of Computation';
```

Part 2 is:

```
SELECT LastName, FirstName, SID
From Student
WHERE SID IN
        (SELECT StudentID
        FROM Enrolled, Course
        WHERE CourseID = CID AND
                CourseName = "Theory of Computation");
```

This new approach also allows us to solve a problem we couldn't earlier: How do we find all the students who have not taken "Theory of Computation"?

We simply change the IN to NOT IN above!

Exercise: What is the difference between the two queries below?

```
SELECT LastName, FirstName, SID
FROM Student
WHERE SID NOT IN
        (SELECT StudentID
        FROM MemberOf, StudentGroup
        WHERE Name = GroupName AND
                Name IN ('HerCTI'));

SELECT LastName, FirstName, SID
FROM Student
WHERE SID IN
        (SELECT StudentID
        FROM MemberOf, StudentGroup
        WHERE Name = GroupName AND
                Name NOT IN ('HerCTI'));
```

There is another SQL statement that is equivalent to **testing whether a set is not empty**. It is called EXISTS.

The following lists the students who have taken "Theory of Computation":

```
SELECT LastName, FirstName, SID
FROM Student
WHERE EXISTS
        (SELECT StudentID
        FROM Enrolled, Course
        WHERE CourseID = CID AND
        StudentID = SID AND
        CourseName = 'Theory of Computation');
```

For each student, we check that there is a record of that student being enrolled in "Theory of Computation". "That student" is expressed using StudentID = SID.

Because we are using a fieldname (SID) from an outer query within an inner query, it is called a **correlated query**.

SQL also offers NOT EXISTS which is equivalent to **testing whether a set is empty**.

Example: Find all student groups with no members

To find the members of a particular student group (say HerCTI) we would write:

```
SELECT StudentID
FROM MemberOf
WHERE Groupname = 'HerCTI';
```

Now we go through all the groups to find which one(s) cause this to be empty.

```
SELECT Name
FROM Studentgroup
WHERE NOT EXISTS
        (SELECT StudentID
        FROM MemberOf
        WHERE Groupname = Name);
```

Union and intersection

There are more set theory definitions that we need to discuss.

The **union** $A \cup B$ of two sets A and B is the set that contains all elements contained by either A or B (or, possibly, both).

Examples:

- {Charlie, Djengo, Hugin, Marcel} ∪ {Joon, Simone}
- {1, 2, 3, …} = {1, 3, 5, …} ∪ {2, 4, 6, …}

Using logic, we can give a more formal definition of the union:

$A \cup B = \{e : e \in A \vee e \in B\}$

If the two sets A and B do not have any elements in common, we say that they are **disjoint**. In that case $A \cup B$ is called the **disjoint union** of A and B.

Example: C = {m : m is a male cat who belonged to Amber} ∪ {f : f is a female cat who belonged to Amber}

Not all unions are disjoint.

Example: {Djengo, Hugin, Simone} ∪ {Djengo, Joon} = {Djengo, Hugin, Joon, Simone}

If the union of two sets is not disjoint, then they must have at least one element in common.

The **intersection** $A \cap B$ is the set that contains all elements that belong to both A and B.

More formally, $A \cap B = \{x : x \in A \wedge x \in B\}$.

Examples:

- {Djengo, Hugin, Simone} ∩ {Djengo, Joon} = {Djengo}
- {Djengo, Hugin, Simone} ∩ {Djengo, Marcel, Charlie, Hugin} = {Djengo, Hugin}

If two sets do not have any elements in common, then their intersection is the empty set.

So two sets A and B are disjoint if $A \cap B = \phi$

Basic set operations in databases

Each of the set operations we have seen so far can be implemented (in some fashion) in SQL.

Union in SQL

Union is implemented in most database engines as UNION.

Example: Find all of the students majoring in computer science and all the student majoring in information systems.

Previously we did this as an OR of two equality conditions in the WHERE clause of an SQL query.

Another way to think about this is as the union of the set of computer science students and the set of information systems students

```
(SELECT LastName, FirstName, SID
FROM Student
WHERE Program = 'COMP-SCI')
UNION
(SELECT LastName, FirstName, SID
FROM Student
WHERE Program = 'INFO-SYS');
```

To take the UNION of two SELECT statements, they have to **return the same type of table**.

Example: It is NOT legal to write:

```
(SELECT LastName
FROM Student
WHERE Program = 'COMP-SCI')
UNION
(SELECT LastName, FirstName, SID
FROM Student
WHERE Program = 'INFO-SYS');
```

Intersection in SQL

Intersection is implemented in a much smaller number of engines (e.g. Microsoft Access does not implement it although H2 does). If it is supported, it is typically called INTERSECT.

Example: List all graduate students in computer science.

Version 1:

```
(SELECT LastName, FirstName, SID
FROM Student
WHERE Program = 'COMP-SCI')
INTERSECT
(SELECT LastName, FirstName, SID
FROM Student
WHERE Career = 'GRD');
```

This is not the best way of writing this. We only want to do operations on primary keys. A better way of writing it is given below.

Version 2:

```
SELECT LastName, FirstName, SID
FROM Student
WHERE SID IN
    (SELECT SID
    FROM Student
    WHERE Program = 'COMP-SCI'
    INTERSECT
    SELECT SID
    FROM Student
    WHERE Career = 'GRD');
```

In databases that do not support intersection, we can eliminate its use through propositional logic.

Example: The query equivalent to the last example would be:

```
SELECT LastName, FirstName, SID
FROM Student
WHERE SID IN
        (SELECT SID
        FROM Student
        WHERE Program = 'COMP-SCI') AND
        SID IN
        (SELECT SID
        FROM Student
        WHERE Career = 'GRD');
```

Exercise: List students who are presidents of some student group and are enrolled in a CSC course. Hint: Do it as an intersection.

More set theory results

The following are some basic results about the set theory we have seen so far.

Lemma: The following statements are true for all sets A and B:

i. $A \cap B = B \cap A$ and $A \cup B = B \cup A$
ii. $A \cap B \subseteq A$
iii. $A \subseteq A \cup B$
iv. If $A \subseteq B$, then $A \cap B = A$ and $A \cup B = B$
v. If $A \cup B \subseteq A$ then $B \subseteq A$; also if $A \subseteq A \cap B$, then $A \subseteq B$

Before we try to prove any of these, it is a good idea to try some examples.

Why? Because until you have looked at some examples, you may not have a sense of what the problem is saying. Also, you may discover that it's not true, and you don't want to waste your time trying to prove something that is false. Above all else, we want to listen to Maass who says to try special cases.

Part (i): Suppose A = {Djengo, Hugin, Simone} and B = {Simone, Hugin, Joon}. Then $A \cap B$ = {Hugin, Simone} = $B \cap A$

Proof (parts ii, iii, and iv are left as exercises):
(i)

Let $x \in A \cap B$. Then x is an element of A and an element of B. But then $x \in B \cap A$.

Alternatively we can write:
$A \cap B = \{x : x \in A \wedge x \in B\}$ by the definition of $\cap$
$= \{x : x \in B \wedge x \in A\}$ since $p \wedge q$ is equivalent to $q \wedge p$
$= B \cap A$ by the definition of $\cap$

How would we prove the second half of this part?

This proof seems obvious. That's because the result is obvious, if you look at it the right way. Draw the Venn diagrams of $A \cap B$.

(v)

Assume that $A \cup B \subseteq A$. This means that any element contained in either A or B has to belong to A. That means in particular that any element in B has to belong to A. So $B \subseteq A$.

Difference and complement

The union and intersection of a set are related to the OR and AND we saw in the propositional logic part of this course. Is there any set theoretical concept that relates to the negation?

The **complement** of a set is the set of all elements not belonging to the set.

More formally, $\overline{A} = \{x : x \notin A\}$

Example: $\overline{\{\text{Djengo}\}}$ is not just the set of all other cats who have belonged to me but all other cats that have ever lived, and all people, and all natural numbers, and the entire rest of the universe.

To make things clearer, set theorists assume that complementation occurs in the "universe of discourse".

Examples:

- $\overline{\{1, 3, 5, 7, ...\}} = \{2, 4, 6, 8, \ldots\}$ not everything in the entire universe except the odd numbers since the university of discourse is assumed to be the natural numbers (e.g. integers greater than 0).
- $\overline{\{\text{Djengo}\}}$ = {Charlie, Hugin, Joon, Marcel, Simone} where the universe of discourse is the cats I've lived with as an adult

Lemma:

i. $A \cap \overline{A} = \phi$
ii. $\overline{A} \cap \overline{B} = \overline{A \cup B}$
iii. $\overline{A} \cup \overline{B} = \overline{A \cap B}$

Proof: We can use propositional logic to prove parts (ii) and (iii).

Part (ii): $\overline{A} \cap \overline{B} = \{x : x \notin A \wedge x \notin B\}$
$= \{x : \overline{x \in A} \wedge \overline{x \in B}\}$

$= \{x: \overline{x \in A \vee x \in B}\}$ by DeMorgan's Law
$= \overline{\{x : x \in A \vee x \in B\}}$ by definition of the complement
$= \overline{A \cup B}$

The **difference** of two sets A and B is the elements that remain in A when we remove everything found in B.

More formally, $A - B = \{x: x \in A \wedge x \notin B\}$.

Note that $A - B = A \cap \overline{B}$.

Lemma: $A \subseteq B$ if and only if $A - B = \phi$

Proof:

If $A \subseteq B$ then all elements of A are in the set B. This means there are no elements of A that are not in B. So the intersection of A and $\overline{B}$ is empty, meaning that $A - B = \phi$.

If $A - B = \phi$, there are no elements belonging to both A and $\overline{B}$. So any element belonging to A cannot belong to $\overline{B}$ which means that it must belong to B. So $A \subseteq B$.

Pairs and tuples

A **pair** of two objects x and y is written as (x, y).

The main characteristic of a pair is that it is **ordered**: a pair has a first component and a second component.

Thus two pairs (x, y) and (u, v) are **equal** when x = u and y = v.

Exercise: Why is (x, y) not the same as {x, y}?

Given two sets X and Y we can form the **Cartesian product** of the two sets by collecting all pairs:

$X \times Y = \{ (x, y) : x \in X \wedge y \in Y\}$

Example: Let X = {Djengo, Hugin} and Y = {Simone, Joon}.

Then X x Y = {(Djengo, Simone), (Djengo, Joon), (Hugin, Simone), (Hugin, Joon)}

We can take the product of more than two sets by building **tuples** instead of pairs.

Example: $X \times Y \times Z = \{(x, y, z) : x \in X \wedge y \in Y \wedge z \in Z\}$ is the set of all triples with first component in X, second component in Y, and third component in Z.

In general, if we have sets $X_1, X_2, \ldots, X_n$, their Cartesian product is:
$X_1 \times X_2 \times \ldots \times X_n = \{(x_1, x_2, \ldots, x_n) : x_1 \in X_1 \wedge x_2 \in X_2 \wedge \ldots \wedge x_n \in X_n\}$

Tuples have the same defining characteristic as pairs:
$(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_n)$ if and only if $x_i = y_i$ for $1 \leq i \leq n$.

Power set

The **power set** is the set of all subsets of a set.

More formally, $P(A) = \{X : X \subseteq A\}$.

Example: With B = {Djengo, Simone}, we have
$P(B) = \{\phi, \{\text{Djengo}\}, \{\text{Simone}\}, \{\text{Djengo, Simone}\}\}$

The empty set and the entire set A always appear in the power set P(A) since they are the trivial subsets of the set.

Exercise: Compute the power sets of the following sets

- {Djengo, Simone, Hugin}
- ϕ

What is the size of the power set of S in relation to the size of S?

$|P(S)| = 2^{|S|}$ for finite sets S

Why?

Subsets differ by which elements they contain, and each element can either be contained or not.
This gives us two choices per element, yielding: 2 * 2 * … * 2 (|S| times) = $2^{|S|}$

Since $2^n > n$ for all n, the power set of a set is always larger than the set itself. (This is also true for infinite sets, as we will see later).

Exercises:

1. How many elements does P({Simone, Hugin, Joon, Djengo, Marcel}) have?
2. Compute P({1, 2, 3}) and draw the result as a diagram which shows how the subsets are included in each other. Start with the empty set at the bottom, {1, 2, 3} at the top. Draw an arrow from one subset A to another B if $A \subseteq B$.