Recap of Feb 11: SQL

- "*" is an abbreviation for all attributes in the `from` list
- implicit tuple variables; table names as tuple variables
- Set operators: `union, intersect, minus, contains, exists, in, not`
- `Insert` command and semantics
- `Delete` command and semantics
- `Update` command and semantics
- Aggregate functions: `avg, min, max, sum, count`
- `group by` clause
- `having` clause
SQL: Multiple Group Bys

Example: using relation  \texttt{emp} (ss\#, ename, dept, cat, sal)

Count the employees and average monthly salary for each employee category in each department

\begin{verbatim}
select    dept, cat, count(*), \text{avg}(sal)/12
from      emp
group by  dept, cat
\end{verbatim}
SQL: Multiple Group Bys

Select … from emp
  group by cat

Select … from emp
  group by dept
SQL: Multiple Group By

Select … from emp
group by dname, cat

note that some dname/cat groups are empty.
SQL: Examples on Having

Find the average salary of employees under 30 for each department with more than 10 such employees

```
select dname, avg(sal) from emp where age<30 group by dname having count(*) > 10
```

SQL: Examples on Having

Find the average salary of employees under 30 for each department with more than 10 employees

```
select e.dname, avg(e.sal)
from emp e
where e.age<30
(group by department)
having 10<any
(select count(ee.ename)
from emp ee
where ee.dname=e.dname)
(why is this query different from the previous one?)
```
Find categories of employees whose average salary exceeds that of programmers

```sql
select cat, avg(sal)
from emp
group by cat
having avg(sal) > (select avg(sal)
                   from emp
                   where cat = 'programmer')
```
SQL: Examples on Having

Find all departments with at least two clerks

```sql
select dname
from emp
where job="clerk"
group by dname
having count(*) >= 2
```
SQL: Examples

Find the names of sailors with the highest rating

```
select sname
from sailors
where rating = (select max(rating)
                from sailors)
```
For each boat, find the number of sailors of rating >7 who have reserved this boat

```
select bid, bname, count(s.sid)
from sailors s, boats b, reserve r
where s.sid=r.sid and r.bid=b.bid and rating>7
group by b.bid
```
SQL: Examples

For each red boat, find the number of sailors who have reserved this boat

```sql
select bid, bname, count(s.sid) 
from sailors s, boats b, reserve r 
where s.sid=r.sid and r.bid=b.bid 
group by b.bid 
having colour="red"
```
SQL: Examples

Difference between the last two queries?

- First one gave a qualification on the tuples
  (take all tuples of the multijoin
discard tuples that do not fulfill ratings>7
then group them by boat id
then find the cardinality of each group)

- Second one gave a qualification for the groups
  (take all tuples of the multijoin
group them by boat id
discard groups representing boats that are non-red
find the cardinality of remaining groups)
And Now, For Something Completely Different...

• The recent SQL material largely covers chapter 4, at least sections 4.1 through 4.6 and some of 4.9.

• Earlier we examined Relational Algebra, covering sections 3.1 through 3.3

• Now we leave chapter 4 and head back to examine sections 3.6 and 3.7, covering *Relational Calculi*
  – based upon predicate calculus
  – non-procedural query languages (descriptive rather than prescriptive)
  – we will examine two relational calculi: tuple calculus and domain calculus
Tuple Calculus

Query: \{ t \mid P(t) \}

- students in CMSC 424
  - \{ t \mid t \text{ enroll } t[\text{course#}] = \text{CMSC424} \}
- students in CMSC 424 conforming with the CMSC-420 prerequisite
  - \{ t \mid t \text{ enroll } s \text{ enroll } t[\text{course#}] = \text{CMSC424} \\
    s[\text{course#}] = \text{CMSC420} \text{ t[ss#] = s[ss#]} \}
Tuple Calculus

- **Quantifiers and free variables**
  - \(? , ?\) quantify the variables following them, binding them to some value. (in the previous slide, s was bound by ?)
  - A tuple variable that is not quantified by ? or ? is called a free variable. (in the previous slide, t was a free variable)

- **Atoms**
  - \(R(t)\) where t is a tuple variable
  - \(t[x] ? s[y]\) where t,s are tuple variables and \(? , ?\) \{?, ?, ?, ?, ?, ?, ?\}
Tuple Calculus

• Formulas
  – an Atom is a Formula
  – If P and Q are Formulas, then so are (P), ? P, P? Q, P? Q, and P? Q
  – If P(t) is a Formula, then so are ?t P(t) and ? t P(t)

• Equivalences
  – ? t P(t) ? ? (? t (? P(t)))
  – ? t P(t) ? ? (? t (? P(t)))
Tuple Calculus

• Safety
  – Math is too powerful; we can easily phrase expressions that describe infinite sets
    \{t \mid t \not\in enroll\}
  – These expressions are called unsafe
  – When we are dealing with finite sets, unsafe expressions happen in expressions that involve negation (\not\in)
  – We can avoid this problem by using an entirely positive (non-negated) scope as the first operand of any conjunction where we use negation. The first operand establishes the scope and the second one filters the established scope.
    \{t \mid t \not\in enroll \land t[course\#] \not\in CMSC-420\}
Domain Calculus

• Another form of relational calculus
• Uses domain variables that take values from an attribute’s domain, rather than values representing an entire tuple
• Closely related to tuple calculus
• Domain Calculus serves as the theoretical basis for the query language QBE, just as the relational algebra we examined earlier forms the basis for SQL
• Expressions are of the form:

\[ \{ < x_1, x_2, x_3, ..., x_n> \mid P( x_1, x_2, x_3, ..., x_n) \} \]
Domain Calculus

- **Atoms**
  - \(< x_1, x_2, x_3, ..., x_n > \) \( ? \) \( R \)
  - \( x \) \( ? \) \( y \) where \( x, y \) are domain variables and
    \( ? \) \( ? \) \{?, ?, ?, ?, ?, ?, ?\}
  - \( x \) \( ? \) \( c \) where \( c \) is a constant

- **Formulas**
  - an atom is a formula
  - If \( P \) and \( Q \) are formulas, then so are \( (P) \), \( ? P \), \( P? Q \), \( P? Q \), and \( P? Q \)
  - If \( P(x) \) is a formula and \( x \) is a domain variable, then \( ?x P(x) \) and
    \( ?x P(x) \) are also formulas
Domain Calculus

• Queries are of the form:
  \{ <x_1, x_2, x_3, ..., x_n> \mid P(x_1, x_2, x_3, ..., x_n) \}

• Examples

  \{ <ss\#, course\#, semester> \mid \text{Enroll}(ss\#, course\#, semester) \}

  \{ <x, y, z> \mid \text{Enroll}(x, y, z) \？ y = \text{CMSC-424} \}
Reductions of Relational Algebra and Calculi

- Relational Algebra (sections 3.2-3.5), Tuple Calculus (section 3.6), and Domain Calculus (section 3.7) can be reduced to each other: they have equivalent expressive power. For every expression in one, we can compute an equivalent expression in the others.
Functional Dependencies

- Important concept in differentiating good database designs from bad ones
- FD is a generalization of the notion of keys
- An FD is a set of attributes whose values uniquely determine the values of the remaining attributes.

Emp(\textit{eno}, \textit{ename}, \textit{sal})
key FDs: \textit{eno} => \textit{ename}

Dept(\textit{dno}, \textit{dname}, \textit{floor})
\textit{eno} => \textit{sal}

Works-in(\textit{eno}, \textit{dno}, \textit{hours})
(\textit{eno, dno}) => \textit{hours}
dno => \textit{dname}
dno => \textit{floor}
Functional Dependencies

- If \( R \) and \( S \) are relations, then \( \text{if } R \subseteq S \text{ holds in the extension } r(R) \text{ of } R \text{ iff for any pair } t_1 \text{ and } t_2 \text{ tuples of } r(R) \text{ such that } t_1(?)=t_2(?) \), then it is also true that \( t_1(?) = t_2(?) \)

- We can use FDs as
  - constraints we wish to enforce (e.g., keys)
  - for checking to see if the FDs are satisfied within the database

<table>
<thead>
<tr>
<th>R( A B C D)</th>
<th>A =&gt; B satisfied?</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2 1 2</td>
<td>A =&gt; C satisfied?</td>
<td>yes</td>
</tr>
<tr>
<td>2 2 2 2</td>
<td>C =&gt; A satisfied?</td>
<td>no</td>
</tr>
<tr>
<td>2 3 2 3</td>
<td>AB =&gt; D satisfied?</td>
<td>yes</td>
</tr>
<tr>
<td>3 3 2 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Functional Dependencies

- Trivial dependencies: $? \Rightarrow ?$
  $? \Rightarrow ?$ if $? ? ?$

- Closure
  - we need to consider all FDs
  - some are implied by others; e.g., FDs are transitive; if $A \Rightarrow B$ and $B \Rightarrow C$, then $A \Rightarrow C$
  - Given $F =$ set of FDs, we want to find $F'$ (the closure of all FDs logically implied by $F$)
Armstrong’s Axioms

- Reflexivity if $? ? ?$ then $? => ?$
- Augmentation if $? => ?$ then $?? ?? ? ??$
- Transitivity if $? => ?$ and $? => ?$ then $? => ?$

Armstrong’s Axioms can be used to derive three additional useful rules:

- Union rule if $? => ?$ and $? => ?$ then $? => ??$
- Decomposition rule if $? => ??$ then $? => ?$ and $? => ?$
- Pseudotransitivity rule if $? => ?$ and $?? => ?$ then $?? => ?$
FD Example

R(A, B, C, G, H, I)

\[ F= (A => B, A => C, CG => H, CG => I, B => H) \]

\[ F+= (A => H, transitivity: A => B => H, CG => HI, union rule: CG => H, CG => I, AG => I, augmentation: A=\to C, AG \to CG \to I, AG => H, augmentation: A=\to C, AG \to CG \to H) \]
Closure of Attribute Sets

- Useful to test if an attribute set is a superkey
- the closure $A^+$ of a set of attributes $A$ under $F$ is the set of all attributes that are functionally determined by $A$
- there is an algorithm to compute the closure

$R(A, B, C, G, H, I)$  
$F=(A=>B, A=>C, CG => H, CG => I, B => H )$

Example: Algorithm to compute $(AG)^+$

- starts with $result=(AG)$
- $A=>B$ expands $result=(AGB)$
- $A=>C$ expands $result=(AGBC)$
- $CG=>H$ expands $result=(AGBCH)$
- $CG=>I$ expands $result=(AGBCHI)$
- $B=>H$ causes no more expansion
Uses of Attribute Closure

• Testing for superkey
  – to test if ? is a superkey, compute ?\(^+\) and determine if ?\(^+\) contains all attributes of R

• Testing functional dependencies
  – to check if a functional dependency ? => ? holds, (is in F+) just check to see if ? ? ?\(^+\)
  – in other words, we compute ?\(^+\) using attribute closure, and then check if the result contains ?.
  – Simple, cheap, and useful test.