



Lecture 9 of 42

Relational Calculus Notes: MP2 Questions

Wednesday, 13 September 2006

William H. Hsu
Department of Computing and Information Sciences, KSU

KSOL course page: <http://snipurl.com/va60>
Course web site: <http://www.kddresearch.org/Courses/Fall-2006/CIS560>
Instructor home page: <http://www.cis.ksu.edu/~bhsu>

Reading for Next Class:
Rest of Chapter 5, Silberschatz *et al.*, 5th edition
JDBC Primer (to be posted on Handouts page)



ODBC Code

- int ODBCexample()

```
{  
    RETCODE error;  
    HENV  env; /* environment */  
    HDBC  conn; /* database connection */  
    SQLAllocEnv(&env);  
    SQLAllocConnect(env, &conn);  
    SQLConnect(conn, "aura.bell-labs.com", SQL_NTS, "avi", SQL_NTS,  
               "avipasswd", SQL_NTS);  
    { .... Do actual work ... }  
  
    SQLDisconnect(conn);  
    SQLFreeConnect(conn);  
    SQLFreeEnv(env);  
}
```





JDBC Code

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection( "jdbc:oracle:thin:@aura.bell-
labs.com:2000:bankdb", userid, passwd);
        Statement stmt = conn.createStatement();
        ... Do Actual Work ....
        stmt.close();
        conn.close();
    }
    catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle);
    }
}
```



Procedural Extensions and Stored Procedures

- SQL provides a **module** language
 - * Permits definition of procedures in SQL, with if-then-else statements, for and while loops, etc.
 - * more in Chapter 9
- Stored Procedures
 - * Can store procedures in the database
 - * then execute them using the **call** statement
 - * permit external applications to operate on the database without knowing about internal details
- These features are covered in Chapter 9 (Object Relational Databases)





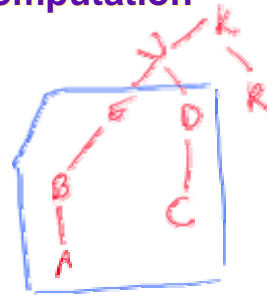
The Power of Recursion

- Recursive views make it possible to write queries, such as transitive closure queries, that cannot be written without recursion or iteration.
 - * Intuition: Without recursion, a non-recursive non-iterative program can perform only a fixed number of joins of *manager* with itself
 - ⇒ This can give only a fixed number of levels of managers
 - ⇒ Given a program we can construct a database with a greater number of levels of managers on which the program will not work
 - * The next slide shows a *manager* relation and each step of the iterative process that constructs *empl* from its recursive definition. The final result is called the *fixed point* of the recursive view definition.
- Recursive views are required to be *monotonic*. That is, if we add tuples to *manager* the view contains all of the tuples it contained before, plus possibly more



Example of Fixed-Point Computation

<i>employee_name</i>	<i>manager_name</i>
Alon	Barinsky
Barinsky	Estovar
Corbin	Duarte
Duarte	Jones
Estovar	Jones
Jones	Klinger
Rensal	Klinger



<i>Iteration number</i>	<i>Tuples in empl</i>
0	
1	(Duarte), (Estovar)
2	(Duarte), (Estovar), (Barinsky), (Corbin)
3	(Duarte), (Estovar), (Barinsky), (Corbin), (Alon)
4	(Duarte), (Estovar), (Barinsky), (Corbin), (Alon)





Chapter 5: Other Relational Languages

- Tuple Relational Calculus
- Domain Relational Calculus
- Query-by-Example (QBE)
- Datalog



Tuple Relational Calculus

- A nonprocedural query language, where each query is of the form $\{t \mid P(t)\}$
- It is the set of all tuples t such that predicate P is true for t
- t is a *tuple variable*, $t[A]$ denotes the value of tuple t on attribute A
- $t \in r$ denotes that tuple t is in relation r
- P is a *formula* similar to that of the predicate calculus





Predicate Calculus Formula

1. Set of attributes and constants
2. Set of comparison operators: (e.g., $<$, \leq , $=$, \neq , $>$, \geq)
3. Set of connectives: and (\wedge), or (\vee), not (\neg)
4. Implication (\Rightarrow): $x \Rightarrow y$, if x is true, then y is true

$$x \Rightarrow y \equiv \neg x \vee y$$

5. Set of quantifiers:

- ▶ $\exists t \in r(Q(t)) \equiv$ "there exists" a tuple in t in relation r such that predicate $Q(t)$ is true
- ▶ $\forall t \in r(Q(t)) \equiv$ Q is true "for all" tuples t in relation r

→ 2 + r . Q(t)



Banking Example

- *branch* (*branch_name*, *branch_city*, *assets*)
- *customer* (*customer_name*, *customer_street*, *customer_city*)
- *account* (*account_number*, *branch_name*, *balance*)
- *loan* (*loan_number*, *branch_name*, *amount*)
- *depositor* (*customer_name*, *account_number*)
- *borrower* (*customer_name*, *loan_number*)



Example Queries

- Find the *loan_number*, *branch_name*, and *amount* for loans of over \$1200

$$\{t \mid t \in \text{loan} \wedge t[\text{amount}] > 1200\}$$

- Find the loan number for each loan of an amount greater than \$1200

$$\{t \mid \exists s \in \text{loan} (t[\text{loan_number}] = s[\text{loan_number}] \wedge s[\text{amount}] > 1200)\}$$

Notice that a relation on schema [*loan_number*] is implicitly defined by the query



Example Queries

- Find the names of all customers having a loan, an account, or both at the bank

$$\{t \mid \exists s \in \text{borrower} (t[\text{customer_name}] = s[\text{customer_name}]) \vee \exists u \in \text{depositor} (t[\text{customer_name}] = u[\text{customer_name}])\}$$

- Find the names of all customers who have a loan and an account at the bank

$$\{t \mid \exists s \in \text{borrower} (t[\text{customer_name}] = s[\text{customer_name}]) \wedge \exists u \in \text{depositor} (t[\text{customer_name}] = u[\text{customer_name}])\}$$



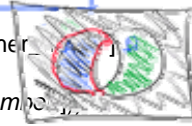


Example Queries

- Find the names of all customers having a loan at the Perryridge branch

$$\{t \mid \exists s \in \text{borrower} (t[\text{customer_name}] = s[\text{customer_name}] \wedge \exists u \in \text{loan} (u[\text{branch_name}] = \text{"Perryridge"} \wedge u[\text{loan_number}] = s[\text{loan_number}])))\}$$

- Find the names of all customers who have a loan at the Perryridge branch, but no account at any branch of the bank

$$\{t \mid \exists s \in \text{borrower} (t[\text{customer_name}] = s[\text{customer_name}] \wedge \exists u \in \text{loan} (u[\text{branch_name}] = \text{"Perryridge"} \wedge u[\text{loan_number}] = s[\text{loan_number}]) \wedge \text{not } \exists v \in \text{depositor} (v[\text{customer_name}] = t[\text{customer_name}]))\}$$


Example Queries

- Find the names of all customers having a loan from the Perryridge branch, and the cities in which they live

$$\{t \mid \exists s \in \text{loan} (s[\text{branch_name}] = \text{"Perryridge"} \wedge \exists u \in \text{borrower} (u[\text{loan_number}] = s[\text{loan_number}] \wedge t[\text{customer_name}] = u[\text{customer_name}]) \wedge \exists v \in \text{customer} (u[\text{customer_name}] = v[\text{customer_name}] \wedge t[\text{customer_city}] = v[\text{customer_city}])))\}$$



Example Queries

- Find the names of all customers who have an account at all branches located in Brooklyn:

$$\{t \mid \exists r \in \text{customer} (t[\text{customer_name}] = r[\text{customer_name}]) \wedge (\forall u \in \text{branch} (u[\text{branch_city}] = \text{"Brooklyn"} \rightarrow \exists s \in \text{depositor} (t[\text{customer_name}] = s[\text{customer_name}] \wedge \exists w \in \text{account} (w[\text{account_number}] = s[\text{account_number}] \wedge (w[\text{branch_name}] = u[\text{branch_name}])))))\}$$

Handwritten annotations: "customer" is circled in green, "Brooklyn" is boxed in green, and the entire expression is enclosed in a blue box. There are also some scribbles and a small "B" in a box.

$$\pi_{CN} \left[\left[(D \bowtie A) \div \pi_{CN} (\sigma_{B=C} (B)) \right] \bowtie C \right]$$


Safety of Expressions

- It is possible to write tuple calculus expressions that generate infinite relations.
- For example, $\{t \mid \neg t \in r\}$ results in an infinite relation if the domain of any attribute of relation r is infinite
- To guard against the problem, we restrict the set of allowable expressions to safe expressions.
- An expression $\{t \mid P(t)\}$ in the tuple relational calculus is *safe* if every component of t appears in one of the relations, tuples, or constants that appear in P
 - * NOTE: this is more than just a syntax condition.
 - ⇒ E.g. $\{t \mid t[A] = 5 \vee \text{true}\}$ is not safe --- it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in P .



Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus
- Each query is an expression of the form:

$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid P(x_1, x_2, \dots, x_n) \}$$

- * x_1, x_2, \dots, x_n represent domain variables
- * P represents a formula similar to that of the predicate calculus



Example Queries

- Find the *loan_number*, *branch_name*, and *amount* for loans of over \$1200

$$\{ \langle l, b, a \rangle \mid \langle l, b, a \rangle \in \text{loan} \wedge a > 1200 \}$$

- Find the names of all customers who have a loan of over \$1200

$$\{ \langle c \rangle \mid \exists l, b, a (\langle c, l \rangle \in \text{borrower} \wedge \langle l, b, a \rangle \in \text{loan} \wedge a > 1200) \}$$

- Find the names of all customers who have a loan from the Perryridge branch and the loan amount:

$$\triangleright \{ \langle c, a \rangle \mid \exists l (\langle c, l \rangle \in \text{borrower} \wedge \exists b (\langle l, b, a \rangle \in \text{loan} \wedge b = \text{"Perryridge"})) \}$$

$$\triangleright \{ \langle c, a \rangle \mid \exists l (\langle c, l \rangle \in \text{borrower} \wedge \langle l, \text{"Perryridge"}, a \rangle \in \text{loan}) \}$$





Example Queries

- Find the names of all customers having a loan, an account, or both at the Perryridge branch:

$$\{ \langle c \rangle \mid \exists l (\langle c, l \rangle \in \text{borrower} \wedge \exists b, a (\langle l, b, a \rangle \in \text{loan} \wedge b = \text{"Perryridge"})) \vee \exists a (\langle c, a \rangle \in \text{depositor} \wedge \exists b, n (\langle a, b, n \rangle \in \text{account} \wedge b = \text{"Perryridge"})) \}$$

- Find the names of all customers who have an account at all branches located in Brooklyn:

$$\{ \langle c \rangle \mid \exists s, n (\langle c, s, n \rangle \in \text{customer}) \wedge \forall x, y, z (\langle x, y, z \rangle \in \text{branch} \wedge y = \text{"Brooklyn"}) \Rightarrow \exists a, b (\langle x, y, z \rangle \in \text{account} \wedge \langle c, a \rangle \in \text{depositor}) \}$$



Safety of Expressions

The expression:

$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid P(x_1, x_2, \dots, x_n) \}$$

is safe if all of the following hold:

- All values that appear in tuples of the expression are values from $dom(P)$ (that is, the values appear either in P or in a tuple of a relation mentioned in P).
- For every "there exists" subformula of the form $\exists x (P_1(x))$, the subformula is true if and only if there is a value of x in $dom(P_1)$ such that $P_1(x)$ is true.
- For every "for all" subformula of the form $\forall x (P_1(x))$, the subformula is true if and only if $P_1(x)$ is true for all values x from $dom(P_1)$.





Query-by-Example (QBE)

- Basic Structure
- Queries on One Relation
- Queries on Several Relations
- The Condition Box
- The Result Relation
- Ordering the Display of Tuples
- Aggregate Operations
- Modification of the Database



QBE — Basic Structure

- A graphical query language which is based (roughly) on the domain relational calculus
- **Two dimensional syntax** – system creates templates of relations that are requested by users
- Queries are expressed “by example”





QBE Skeleton Tables for the Bank Example

branch	branch-name	branch-city	assets

customer	customer-name	customer-street	customer-city

loan	loan-number	branch-name	amount



QBE Skeleton Tables (Cont.)

branch	branch_name	branch_city	assets

customer	customer_name	customer_street	customer_city

loan	loan_number	branch_name	amount

borrower	customer_name	loan_number

account	account_number	branch_name	balance

depositor	customer_name	account_number





Queries on One Relation

- Find all loan numbers at the Perryridge branch

<i>loan</i>	<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
	P._x	Perryridge	

- _x is a variable (optional; can be omitted in above query)
- P. means print (display)
- duplicates are removed by default
- To retain duplicates use P.ALL

<i>loan</i>	<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
	P.ALL.	Perryridge	



Queries on One Relation (Cont.)

- Display full details of all loans
 - Method 1:

<i>loan</i>	<i>loan-number</i>	<i>branch-name</i>	<i>amount</i>
	P._x	P._y	P._z

- Method 2: Shorthand notation

<i>loan</i>	<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
P.			





Queries on One Relation (Cont.)

- Find the loan number of all loans with a loan amount of more than \$700

<i>loan</i>	<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
	P.		>700

- Find names of all branches that are not located in Brooklyn

<i>branch</i>	<i>branch_name</i>	<i>branch_city</i>	<i>assets</i>
	P.	¬ Brooklyn	



Queries on One Relation (Cont.)

- Find the loan numbers of all loans made jointly to Smith and Jones.

<i>borrower</i>	<i>customer_name</i>	<i>loan_number</i>
	Smith	P. _x
	Jones	¬ _x

- Find all customers who live in the same city as Jones

<i>customer</i>	<i>customer_name</i>	<i>customer_street</i>	<i>customer_city</i>
	P. _x		¬ _y
	Jones		¬ _y



Queries on Several Relations

- Find the names of all customers who have a loan from the Perryridge branch.

<i>loan</i>	<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
	<i>_x</i>	Perryridge	

<i>borrower</i>	<i>customer_name</i>	<i>loan_number</i>
	<i>P._y</i>	<i>_x</i>



Queries on Several Relations (Cont.)

- Find the names of all customers who have both an account and a loan at the bank.

<i>depositor</i>	<i>customer_name</i>	<i>account_number</i>
	<i>P._x</i>	

<i>borrower</i>	<i>customer_name</i>	<i>loan_number</i>
	<i>_x</i>	





Negation in QBE

- Find the names of all customers who have an account at the bank, but do not have a loan from the bank.

<i>depositor</i>	<i>customer_name</i>	<i>account_number</i>
	P. <i>x</i>	

<i>borrower</i>	<i>customer_name</i>	<i>loan_number</i>
\neg	\neg <i>x</i>	

\neg means "there does not exist"



Negation in QBE (Cont.)

- Find all customers who have at least two accounts.

<i>depositor</i>	<i>customer_name</i>	<i>account_number</i>
	P. <i>x</i>	<i>y</i>
	\neg <i>x</i>	\neg <i>y</i>

\neg means "not equal to"





The Condition Box

- Allows the expression of constraints on domain variables that are either inconvenient or impossible to express within the skeleton tables.
- Complex conditions can be used in condition boxes
- Example: Find the loan numbers of all loans made to Smith, to Jones, or to both jointly.

<i>borrower</i>	<i>customer_name</i>	<i>loan_number</i>
	<i>_n</i>	P. <i>_x</i>
<i>conditions</i>		
<i>_n = Smith or _n = Jones</i>		



Condition Box (Cont.)

- QBE supports an interesting syntax for expressing alternative values

<i>branch</i>	<i>branch_name</i>	<i>branch_city</i>	<i>assets</i>
	P.	<i>_x</i>	
<i>conditions</i>			
<i>_x = (Brooklyn or Queens)</i>			





Condition Box (Cont.)

- Find all account numbers with a balance greater than \$1,300 and less than \$1,500

<i>account</i>	<i>account_number</i>	<i>branch_name</i>	<i>balance</i>			
	P.		<i>_x</i>			
<table border="1"> <thead> <tr> <th><i>conditions</i></th> </tr> </thead> <tbody> <tr> <td>$_x \geq 1300$</td> </tr> <tr> <td>$_x \leq 1500$</td> </tr> </tbody> </table>				<i>conditions</i>	$_x \geq 1300$	$_x \leq 1500$
<i>conditions</i>						
$_x \geq 1300$						
$_x \leq 1500$						

- Find all account numbers with a balance greater than \$1,300 and less than \$2,000 but not exactly \$1,500.

<i>account</i>	<i>account_number</i>	<i>branch_name</i>	<i>balance</i>		
	P.		<i>_x</i>		
<table border="1"> <thead> <tr> <th><i>conditions</i></th> </tr> </thead> <tbody> <tr> <td>$_x = (\geq 1300 \text{ and } \leq 2000 \text{ and } \neg 1500)$</td> </tr> </tbody> </table>				<i>conditions</i>	$_x = (\geq 1300 \text{ and } \leq 2000 \text{ and } \neg 1500)$
<i>conditions</i>					
$_x = (\geq 1300 \text{ and } \leq 2000 \text{ and } \neg 1500)$					



Condition Box (Cont.)

- Find all branches that have assets greater than those of at least one branch located in Brooklyn

<i>branch</i>	<i>branch_name</i>	<i>branch_city</i>	<i>assets</i>		
	P. <i>_x</i>	Brooklyn	<i>_y</i>		
			<i>_z</i>		
<table border="1"> <thead> <tr> <th><i>conditions</i></th> </tr> </thead> <tbody> <tr> <td>$_y > _z$</td> </tr> </tbody> </table>				<i>conditions</i>	$_y > _z$
<i>conditions</i>					
$_y > _z$					