



Lecture 12 of 42

Database Design Overview Notes: Relational Calculi, PS3

Wednesday, 20 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:
Section 6.1 – 6.2, Silberschatz *et al.*, 5th edition



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:

1. 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).
2. 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.
3. 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 $dom(P_1)$.



Recursion in SQL

- Starting with SQL:1999, SQL permits recursive view definition
- E.g. query to find all employee-manager pairs

```
with recursive empl (emp, mgr) as (  
    select emp, mgr  
    from manager  
    union  
    select manager.emp, empl.mgr  
    from manager, empl  
    where manager.mgr = empl.emp )  
select *  
from empl
```





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



Query Example

- Find all customers who have an account at all branches located in Brooklyn.
 - * Approach: for each customer, find the number of branches in Brooklyn at which they have accounts, and compare with total number of branches in Brooklyn
 - * QBE does not provide subquery functionality, so both above tasks have to be combined in a single query.
 - ⇒ Can be done for this query, but there are queries that require subqueries and cannot always be expressed in QBE.
- In the query on the next page
 - ▶ CNT.UNQ.ALL_*w* specifies the number of distinct branches in Brooklyn. Note: The variable *w* is not connected to other variables in the query
 - ▶ CNT.UNQ.ALL_*z* specifies the number of distinct branches in Brooklyn at which customer *x* has an account.



Query Example (Cont.)

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

<i>account</i>	<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
	_y	_z	

<i>branch</i>	<i>branch_name</i>	<i>branch_city</i>	<i>assets</i>
	_z	Brooklyn	
	_w	Brooklyn	

<i>conditions</i>
CNT.UNQ._z = CNT.UNQ._w



Modification of the Database – Deletion

- Deletion of tuples from a relation is expressed by use of a D. command. In the case where we delete information in only some of the columns, null values, specified by –, are inserted.
- Delete customer Smith

<i>customer</i>	<i>customer_name</i>	<i>customer_street</i>	<i>customer_city</i>
D.	Smith		

- Delete the *branch_city* value of the branch whose name is "Perryridge".

<i>branch</i>	<i>branch_name</i>	<i>branch_city</i>	<i>assets</i>
	Perryridge	D.	



Deletion Query Examples

- Delete all loans with a loan amount greater than \$1300 and less than \$1500.
 - * For consistency, we have to delete information from loan and borrower tables

<i>loan</i>	<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
D.	<i>-y</i>		<i>-x</i>

<i>borrower</i>	<i>customer_name</i>	<i>loan_number</i>
D.		<i>-y</i>
<i>conditions</i>		
<i>-x = (≥ 1300 and ≤ 1500)</i>		



Deletion Query Examples (Cont.)

- Delete all accounts at branches located in Brooklyn.

<i>account</i>	<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
D.	<i>-y</i>	<i>-x</i>	
<i>depositor</i>			
D.		<i>-y</i>	
<i>branch</i>			
	<i>-x</i>	Brooklyn	



Modification of the Database – Insertion

- Insertion is done by placing the I. operator in the query expression.
- Insert the fact that account A-9732 at the Perryridge branch has a balance of \$700.

<i>account</i>	<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
I.	A-9732	Perryridge	700



Modification of the Database – Insertion (Cont.)

- Provide as a gift for all loan customers of the Perryridge branch, a new \$200 savings account for every loan account they have, with the loan number serving as the account number for the new savings account.

<i>account</i>	<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
I.	_x	Perryridge	200

<i>depositor</i>	<i>customer_name</i>	<i>account_number</i>
I.	_y	_x

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

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



Modification of the Database – Updates

- Use the U. operator to change a value in a tuple without changing *all* values in the tuple. QBE does not allow users to update the primary key fields.
- Update the asset value of the Perryridge branch to \$10,000,000.

<i>branch</i>	<i>branch_name</i>	<i>branch_city</i>	<i>assets</i>
	Perryridge		U.10000000

- Increase all balances by 5 percent.

<i>account</i>	<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
			U..x * 1.05



Microsoft Access QBE

- Microsoft Access supports a variant of QBE called Graphical Query By Example (GQBE)
- GQBE differs from QBE in the following ways
 - * Attributes of relations are listed vertically, one below the other, instead of horizontally
 - * Instead of using variables, lines (links) between attributes are used to specify that their values should be the same.
 - ⇒ Links are added automatically on the basis of attribute name, and the user can then add or delete links
 - ⇒ By default, a link specifies an inner join, but can be modified to specify outer joins.
 - * Conditions, values to be printed, as well as group by attributes are all specified together in a box called the **design grid**





An Example Query in Microsoft Access QBE

- Example query: Find the *customer_name*, *account_number* and *balance* for all accounts at the Perryridge branch

The screenshot shows the Microsoft Access Query Design view. Two tables, 'account' and 'depositor', are displayed. The 'account' table has fields: account_number, branch_name, and balance. The 'depositor' table has fields: customer_name and account_number. A line connects the 'account_number' field in the 'depositor' table to the 'account_number' field in the 'account' table. Below the tables is the Query Design Grid:

Field:	customer_name	account_number	balance	branch_name
Table:	depositor	account	account	account
Sort:				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Criteria:				"Perryridge"
or:				



An Aggregation Query in Access QBE

- Find the *name*, *street* and *city* of all customers who have more than one account at the bank

The screenshot shows the Microsoft Access Query Design view. Two tables, 'customer' and 'depositor', are displayed. The 'customer' table has fields: customer_name, customer_street, and customer_city. The 'depositor' table has fields: customer_name and account_number. A line connects the 'customer_name' field in the 'depositor' table to the 'customer_name' field in the 'customer' table. Below the tables is the Query Design Grid:

Field:	customer_name	customer_street	customer_city	account_number
Table:	customer	customer	customer	depositor
Total:	Group By	Group By	Group By	Count
Sort:				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Criteria:				>1
or:				



Aggregation in Access QBE

- The row labeled **Total** specifies
 - * which attributes are group by attributes
 - * which attributes are to be aggregated upon (and the aggregate function).
 - * For attributes that are neither group by nor aggregated, we can still specify conditions by selecting **where** in the Total row and listing the conditions below
- As in SQL, if group by is used, only group by attributes and aggregate results can be output



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*)

select A, sum r,s where P(A ₁ , A ₂ , A ₃)	$T_{A_1}(\sigma_{P(A_1, A_2, A_3)} r \times s)$	$T \equiv (A_1, \dots)$ $\{t \in T \mid$ $t[A_1] =$ $r[A_1]$ $\wedge P(A_2, A_3)\}$
SQ	Reln Alg	Reln calc.



Chapter 6: Entity-Relationship Model

- Design Process
- Modeling
- Constraints
- E-R Diagram
- Design Issues
- Weak Entity Sets
- Extended E-R Features
- Design of the Bank Database
- Reduction to Relation Schemas
- Database Design
- UML



Modeling

- A *database* can be modeled as:
 - * a collection of entities,
 - * relationship among entities.
- An **entity** is an object that exists and is distinguishable from other objects.
 - * Example: specific person, company, event, plant
- Entities have *attributes*
 - * Example: people have *names* and *addresses*
- An **entity set** is a set of entities of the same type that share the same properties.
 - * Example: set of all persons, companies, trees, holidays





Entity Sets *customer* and *loan*

customer_id	customer_name	customer_street	customer_city	loan_number	amount
321-12-3123	Jones	Main	Harrison	L-17	1000
019-28-3746	Smith	North	Rye	L-23	2000
677-89-9011	Hayes	Main	Harrison	L-15	1500
555-55-5555	Jackson	Dupont	Woodside	L-14	1500
244-66-8800	Curry	North	Rye	L-19	500
963-96-3963	Williams	Nassau	Princeton	L-11	900
335-57-7991	Adams	Spring	Pittsfield	L-16	1300

customer *loan*



Relationship Sets

- A **relationship** is an association among several entities

Example:

Hayes depositor A-102
customer entity relationship set *account* entity

- A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

$$\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$

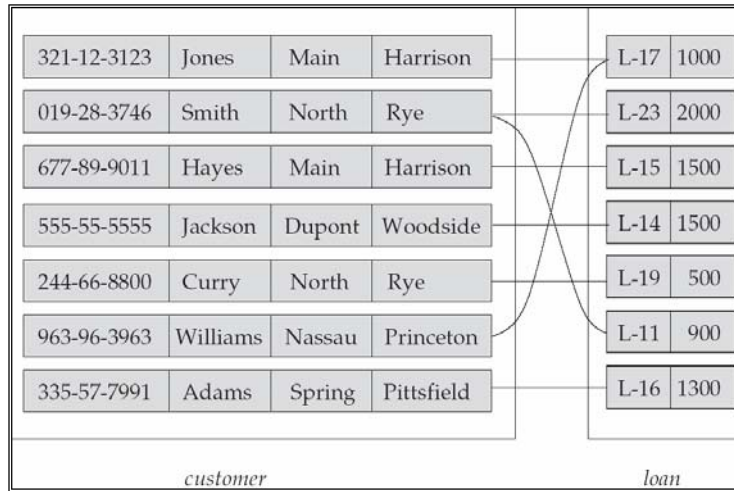
where (e_1, e_2, \dots, e_n) is a relationship

* Example:

$(\text{Hayes}, \text{A-102}) \in \text{depositor}$



Relationship Set *borrower*



Relational Database Design

- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Decomposition Using Functional Dependencies
- Functional Dependency Theory
- Algorithms for Functional Dependencies
- Decomposition Using Multivalued Dependencies
- More Normal Form
- Database-Design Process
- Modeling Temporal Data



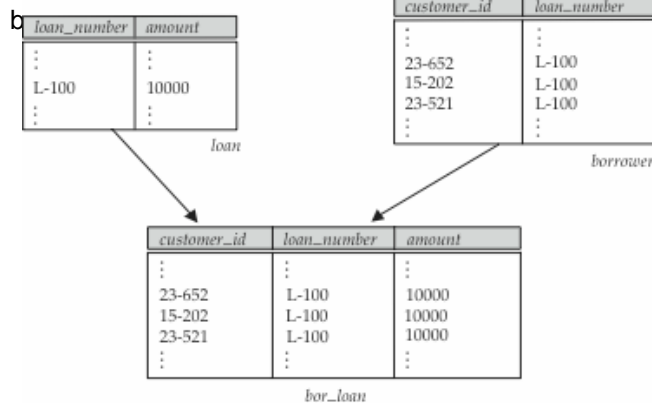
The Banking Schema

- $branch = (branch_name, branch_city, assets)$
- $customer = (customer_id, customer_name, customer_street, customer_city)$
- $loan = (loan_number, amount)$
- $account = (account_number, balance)$
- $employee = (employee_id, employee_name, telephone_number, start_date)$
- $dependent_name = (employee_id, dname)$
- $account_branch = (account_number, branch_name)$
- $loan_branch = (loan_number, branch_name)$
- $borrower = (customer_id, loan_number)$
- $depositor = (customer_id, account_number)$
- $cust_banker = (customer_id, employee_id, type)$
- $works_for = (worker_employee_id, manager_employee_id)$
- $payment = (loan_number, payment_number, payment_date, payment_amount)$
- $savings_account = (account_number, interest_rate)$
- $checking_account = (account_number, overdraft_amount)$



Combine Schemas?

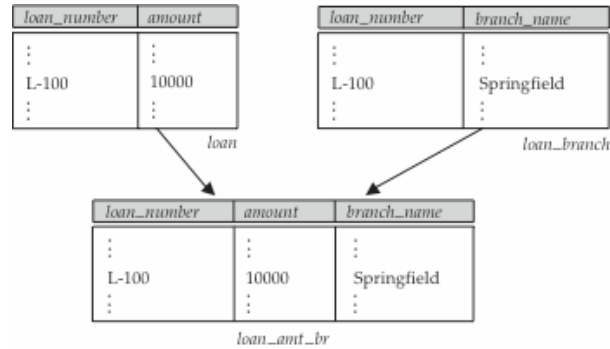
- Suppose we combine $borrow$ and $loan$ to get
 $bor_loan = (customer_id, loan_number, amount)$
- Result is possible repetition of information (L-100 in example)





A Combined Schema Without Repetition

- Consider combining *loan_branch* and *loan*
 $loan_amt_br = (loan_number, amount, branch_name)$
- No repetition (as suggested by example below)

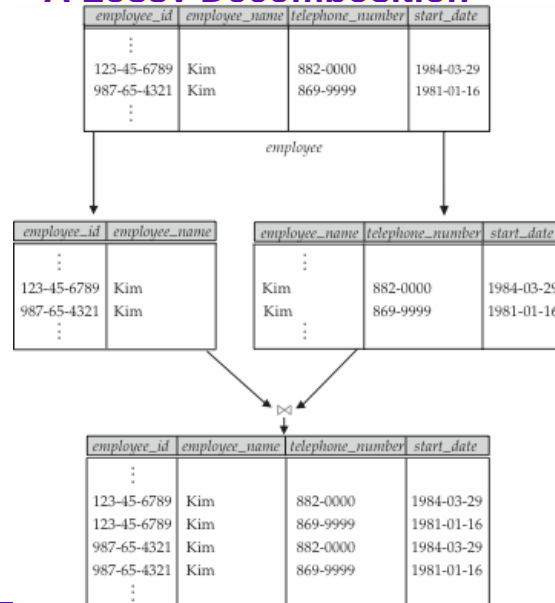


What About Smaller Schemas?

- Suppose we had started with *bor_loan*. How would we know to split up (**decompose**) it into *borrower* and *loan*?
- Write a rule "if there were a schema $(loan_number, amount)$, then *loan_number* would be a candidate key"
- Denote as a **functional dependency**:
 $loan_number \rightarrow amount$
- In *bor_loan*, because *loan_number* is not a candidate key, the amount of a loan may have to be repeated. This indicates the need to decompose *bor_loan*.
- Not all decompositions are good. Suppose we decompose *employee* into
 $employee1 = (employee_id, employee_name)$
 $employee2 = (employee_name, telephone_number, start_date)$
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a lossy decomposition.



A Lossy Decomposition



First Normal Form

- Domain is atomic if its elements are considered to be indivisible units
 - * Examples of non-atomic domains:
 - ⇒ Set of names, composite attributes
 - ⇒ Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in first normal form if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
 - * Example: Set of accounts stored with each customer, and set of owners stored with each account
 - * We assume all relations are in first normal form (and revisit this in Chapter 9)