Theory: Relational Calculus

Textbook (new edition), Section 5.2

Relational Calculus

- An alternative, but equivalent, technique for representing operations on relations

- Employs predicates:
  - Functions that return true or false after evaluating arguments
    - When arguments are instantiated with real values, called a proposition
  - The set of all x for which a given predicate P is true is written as: $\{x \mid P(x)\}$

Tuple Relational Calculus

- We will constrain our predicates to take single arguments, a tuple of data (an entire row of a table)

  {S | Student(S)}

returns the set of tuples S such that Student(S) is true *(… such that S is a member of the Student* relation)

Essentially, the notion of FROM in SQL

Tuple Relational Calculus: Predicates

P(x) must be a formula:

Atoms are an instance of a simple formula

- R(S), where S is a tuple variable and R is a relation
- S.a1 op S.a2, where *op* is a valid comparison operator
- S.a1 op c, where *op* is a valid comparison operator and *c* is a constant

Given a formula, use of the following operators with formula are also formula

AND	&	*Conjunction*
OR	|	*Disjunction*
NEGATION	~	

Tuple Relational Calculus: Predicate Examples

Selection

RC: {X | Student(X) & X.GPA > 3.5}

RA: $\sigma_{GPA > 3.5}$ (Student)

SQL: SELECT * FROM Student WHERE GPA > 3.5;

Selection and Projection

RC: {X.lastName | Student(X) & X.major='CSC')

RA: $\Pi_{lastName}(\sigma_{major='CSC'}(Student))$

SQL: SELECT lastName FROM Student WHERE major='CSC';

Note: Tuple relational calculus does not specify an ordering of operators for retrieving the data as RA and SQL do – purely a statement of what to retrieve

Tuple Relational Calculus: Quantifiers

Quantifiers: Allow statement of how many instances a predicate applies to

- Existential quantifier ($\exists x$): "there exists" → *must be true for at least one instance*
- Universal quantifier ($\forall x$): "for all" → *must be true for all instances*

If F is a formula with variable X, then ($\exists x$)(F) and ($\forall x$)(F) are also formula

Tuple Relational Calculus: Quantifiers

What are the student details for students enrolled in any course this semester?

Provide student information for all students such that there exists at least one entry in the Enrollment relation for that student

RC: $\{S \mid Student(S) \ \& \ (\exists E)(Enrollment(E) \ \& \ E.studentID == S.studentID)\}$

RA: $Student \rhd_{Student.studentID=Enrollment.studentID} Enrollment$

SQL: SELECT s.* FROM Student s, Enrollment e WHERE s.studentID = e.studentID

Tuple Relational Calculus: Quantifiers

- Quantifiers will be particularly useful when representing
  - JOINS, more often than not using $\exists x$

- DeMorgan's Laws (rewriting quantifiers)
  - $(\exists x)(F(x)) \leftrightarrow \sim(\forall x)(\sim(F(x)))$
  - $(\forall x)(F(x)) \leftrightarrow \sim(\exists x)(\sim(F(x)))$
  - $(\exists x)(F1(x) \,\&\, F2(x)) \leftrightarrow \sim(\forall x)(\sim(F1(x)) \mid \sim(F2(x)))$
  - $(\forall x)\ (F1(x) \,\&\, F2(x)) \leftrightarrow \sim(\exists x)(\sim(F1(x)) \mid \sim(F2(x)))$

Tuple Relational Calculus: Practice

- List the price and type of all rooms at the Winston hotel:

  RA: $\Pi_{\text{price,type}}(\text{Room} \rhd_{\text{Room.hotelNumber=Hotel.hotelNumber}} (\sigma_{\text{hotelName='Winston'}} (\text{Hotel})))$

  SQL: SELECT price,type FROM Room NATURAL JOIN (SELECT hotelNumber FROM Hotel WHERE hotelName='Winston' AS WH);

  RC?

  {X | Room(X) & (∃H)(Hotel(H) & (H.hotelNumber==X.hotelNumber) & (H.hotelName=='Winston'))}

Tuple Relational Calculus: Practice

- List all guests details for guests currently staying at the Winston hotel.

RA: $\text{Guest} \rhd_{\text{Guest.guestNumber=Booking.GuestNumber}} (\sigma_{\text{dateFrom} \leq \text{'02-09-12'} \ \&\& \ \text{dateTo} \geq \text{'02-09-12'}} (\text{Booking} \rhd \lhd_{\text{Booking.hotelNumber=Hotel.hotelNumber}} (\sigma_{\text{hotelName = 'Winston'}}(\text{Hotel}))))$

SQL:

```
SELECT g.guestName FROM guest g NATURAL JOIN (SELECT * FROM booking b
NATURAL JOIN (SELECT * FROM hotel WHERE hotelName='Winston Hotel') as w
WHERE dateFrom <= '2012-02-16' AND dateTo >= '2012-02-16') AS z;
```

RC?

{G | Guest(G) & (∃H)(∃B)(Hotel(H) & Booking(B) &
(B.guestNumber==G.guestNumber) & (B.hotelNumber == H.hotelNumber) &
(H.hotelName==‘Winston’) &
(B.dateFrom<=‘2012-02-16’) & (B.dateTo >=‘2012-02-16’))}

Design: Normalization

Normalization: Motivation

Properties of a good database design:

- Minimal number of attributes necessary to support data
- Attributes with close logical relationships in same relation
- Minimal redundancy

Results of employing appropriate designed databases:

- Updates require fewer operations
- Likelihood of introducing inconsistencies is reduced
- Smaller storage (disk/memory) requirements to represent database

Normalization

Bear with me!

While the next few slides may seem obvious & common sense, the notions are important, and there may be subtle applications of the ideas we end up dealing with

Two Representations of Data

- Consider the following relations:

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Every staff member is associated with one branch
staffNo is the primary key (unique identifier)

Both represent the same information

Operation Anomalies

- Using a relation like *BranchStaff* leads to the potential for a number of problems when manipulating data:

- *Insertion anomalies for BranchStaff:*
  - When entering a new staff member, have to add all branch details for branch associated with that staff member
    - Higher likelihood of error than just having to record branchNo
  - New branch information can't be recorded until a staff member is assigned to the branch
    - Otherwise, an entry in this table would have have NULLs on the Staff side, but that violates the entity (primary key) integrity rule

Operation Anomalies

- *Deletion anomalies for BranchStaff:*
  - Removing the last staff member associated with a branch loses all information for that branch

- *Modification anomalies for BranchStaff:*
  - Updating a property of a branch, such as the address, requires updating the tuples for all staff members that work for that branch
    - Higher likelihood of error than just having to update in one place

Two Representations of Data

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

This representation seems better, as none of the anomalies discussed exist here.

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

We need to ensure, if we are decomposing this to what is on the LHS (left-hand-side):

-*Lossless-join:* NO DATA LOSS
Any instance of StaffBranch can be identified in separatedStaff, Branch

-*Dependency preservation:* Any constraints on StaffBranch can be enforced at local level, without having to perform a join.

Update Anomalies

- The following dentist/patient appointment relation is susceptible to update anomalies.
- Suggest 1 each of an insertion, deletion, and update anomaly:

staffNo	dentistName	patNo	patName	appointment date	appointment time	surgeryNo
S1011	Tony Smith	P100	Gillian White	12-Sep-04	10.00	S15
S1011	Tony Smith	P105	Jill Bell	12-Sep-04	12.00	S15
S1024	Helen Pearson	P108	Ian MacKay	12-Sep-04	10.00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sep-04	14.00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Sep-04	16.30	S15
S1032	Robin Plevin	P110	John Walker	15-Sep-04	18.00	S13

A patient is given an appointment at a specific date and time with a dentist located at a particular surgery center.

Update Anomalies

- Insertion: If a patient is added, then **all** the dentist information *(number, name, location)* for the dentist assigned to that patient must be added
- Insertion: A patient must be assigned a dentist before their appointment can be recorded.

staffNo	dentistName	patNo	patName	appointment date	appointment time	surgeryNo
S1011	Tony Smith	P100	Gillian White	12-Sep-04	10.00	S15
S1011	Tony Smith	P105	Jill Bell	12-Sep-04	12.00	S15
S1024	Helen Pearson	P108	Ian MacKay	12-Sep-04	10.00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sep-04	14.00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Sep-04	16.30	S15
S1032	Robin Plevin	P110	John Walker	15-Sep-04	18.00	S13

Update Anomalies

- Delete: Deleting dentist Tony Smith deletes all appointment information for patient Gillian White (instead of her being reassigned)
- Update/modification: If dentist Tony Smith is reassigned to another surgery location for a given day, **all** instances of Tony Smith entries have to be updated for that day

staffNo	dentistName	patNo	patName	appointment date	appointment time	surgeryNo
S1011	Tony Smith	P100	Gillian White	12-Sep-04	10.00	S15
S1011	Tony Smith	P105	Jill Bell	12-Sep-04	12.00	S15
S1024	Helen Pearson	P108	Ian MacKay	12-Sep-04	10.00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sep-04	14.00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Sep-04	16.30	S15
S1032	Robin Plevin	P110	John Walker	15-Sep-04	18.00	S13

Design: Functional Dependencies

Textbook (new edition), Chapter 14 & 15

Understanding how attributes related to each other can support our organizing the data appropriately

Functional Dependencies

- First, an assumption:
  - Assume that the database is represented by a single universal relation R = (A,B,C,…, Z)
    - Implicitly, every attribute has different name
    - One massive table (holds all data)

- A definition:
  - If A and B are attributes of a relation R, B is functionally dependent on A ($A \rightarrow B$) if each value of A is associated with exactly one value of B.

Functional Dependencies

- If A and B are attributes of a relation R, *B is functionally dependent on A (A→B)* if *each value of A is associated with exactly one value of B*.

  - If any tuples share the same value A, then they must also have the same value B. Sharing value for B does not require sharing value for A though!

  - Left hand-side(A) is called the *determinant* (knowing that value determines B)

  - Look for 1:1 multiplicities between attributes (previously we look at multiplicities between entities)
  - Must hold for all possible entries for attributes, not just the entries occurring in the table

Functional Dependencies

- FD: studentID → lastName, studentID → firstName, studentID → year, studentID → major, studentID → GPA
- None of the others hold though (GPA→ major, major → year, major → lastName, etc.)

Student

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

It looks like there will be a functional dependency for attributes with the primary key

Functional Dependencies

- FDs: staffNo → sName, staffNo→position, staffNo→salary, staffNo→branchNo, staffNo→bAddress, branchNo → bAddress, bAddress → branchNo

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

How about position to salary?
Reasonable to consider – if play same role, get same salary?

Does not hold

Functional Dependencies

- Functional dependency statements should be "minimized"
  - Use as few attributes as possible on the LHS

- An attribute B is *fully functionally dependent* on attribute A if B is functionally dependent on A, but not on any proper subset of A

staffNo → position

NOT

staffNo, staffName → position

Functional Dependencies & Keys

- Functional dependencies can *determine primary/candidate keys.*
  - For attribute A to be a candidate key, there must exist a functional dependence for all other attributes on A

FD: studentID → lastName; studentID → firstName; studentID → year; studentID → major; studentID → GPA

Student

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

Functional Dependencies & Keys

- FDs: staffNo → sName, staffNo→position, staffNo→salary, staffNo→branchNo, staffNo→bAddress, branchNo → bAddress, bAddress → branchNo

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

- It is only attribute staffNo for which all other attributes show a functional dependency

Functional Dependencies

- What are some fully functional dependencies in this relation?
  - *I'll start:* staffNo → dentistName

staffNo	dentistName	patNo	patName	appointment date	time	surgeryNo
S1011	Tony Smith	P100	Gillian White	12-Sep-04	10.00	S15
S1011	Tony Smith	P105	Jill Bell	12-Sep-04	12.00	S15
S1024	Helen Pearson	P108	Ian MacKay	12-Sep-04	10.00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sep-04	14.00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Sep-04	16.30	S15
S1032	Robin Plevin	P110	John Walker	15-Sep-04	18.00	S13

Functional Dependencies

- What are some fully functional dependencies in this relation?
  - staffNo → dentistName;
  - patNo → patientName;
  - staffNo, appointmentDate → surgeryNo
  - patNo, date,time →surgeryNo
  - surgeryNo,date,time →staffNo
  - staffNo, patNo, date → surgeryNo // or dentist or patientName
  - …

staffNo	dentistName	patNo	patName	appointment date	time	surgeryNo
S1011	Tony Smith	P100	Gillian White	12-Sep-04	10.00	S15
S1011	Tony Smith	P105	Jill Bell	12-Sep-04	12.00	S15
S1024	Helen Pearson	P108	Ian MacKay	12-Sep-04	10.00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sep-04	14.00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Sep-04	16.30	S15
S1032	Robin Plevin	P110	John Walker	15-Sep-04	18.00	S13

Functional Dependencies

Given analysis of functional dependencies, what is an appropriate primary key for this relation?

staffNo	dentistName	patNo	patName	appointment date	time	surgeryNo
S1011	Tony Smith	P100	Gillian White	12-Sep-04	10.00	S15
S1011	Tony Smith	P105	Jill Bell	12-Sep-04	12.00	S15
S1024	Helen Pearson	P108	Ian MacKay	12-Sep-04	10.00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sep-04	14.00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Sep-04	16.30	S15
S1032	Robin Plevin	P110	John Walker	15-Sep-04	18.00	S13

Functional Dependencies

Given analysis of functional dependencies, what is an appropriate primary key for this relation?

staffNo	dentistName	patNo	patName	appointment date	time	surgeryNo
S1011	Tony Smith	P100	Gillian White	12-Sep-04	10.00	S15
S1011	Tony Smith	P105	Jill Bell	12-Sep-04	12.00	S15
S1024	Helen Pearson	P108	Ian MacKay	12-Sep-04	10.00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sep-04	14.00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Sep-04	16.30	S15
S1032	Robin Plevin	P110	John Walker	15-Sep-04	18.00	S13

staffNo, date, time?

Design: Normalization

Textbook (new edition), Chapter 14 & 15

Normalization

- Analysis of functional dependencies can help us to appropriately decompose relations, a process known as *normalization*

- These decompositions should:
  - Reduce update anomalies
  - Reduce redundancy

- Progressive levels of decomposition we can employ (normal forms)