Invariants Underlying Relational Databases & Queries

Limsoon Wong

Guest Lecture for CS2306S

31 January 2007

What is an invariant?

- **Suppose you have a bag of x red beans and y green beans**
- **Repeat the following:**
  - Remove 2 beans
  - If both green, discard both
  - If both red, discard one, put back one
  - If one green and one red, discard red, put back green
- **If one bean is left behind, can you predict its colour?**

Plan

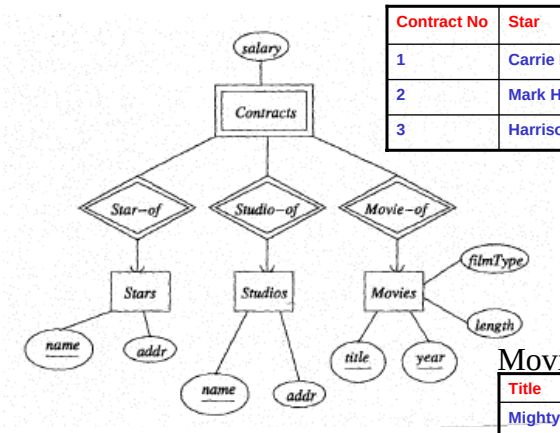
- **Database Design**
  - Relational data model
  - Dependencies
  - Normal forms
- **Transaction Management**
  - Serializable Schedules
  - Two-Phase Locking
- **Query Languages**
  - Relational Algebra
  - Query Optimization
  - Expressive Power
  - Beyond Relational Algebra

Database Design

Relational Data Model

- **Data are represented as a two-dimensional table**
- **It is a logical representation, not a physical representation**
  - Ordering of the rows is irrelevant
  - Ordering of the columns is irrelevant
  - How the rows and columns of a table are stored is irrelevant
  - …

Example



Contracts

Contract No	Star	Studio	Title	Salary
1	Carrie Fisher	Fox	Star Wars	$$$
2	Mark Hamill	Fox	Star Wars	$$$
3	Harrison Ford	Fox	Star Wars	$$$

Stars

Name	Address
Carrie Fisher	Hollywood
Mark Hamill	Brentwood
Harrison Ford	Beverly Hills

Movies

Title	Year	Length	Film Type
Mighty Ducks	1991	104	Color
Wayne's World	1992	95	Color
Star Wars	1977	124	Color

Design Issues

- How many possible alternate ways to represent movies using tables?
- Why this particular set of tables to represent movies?
- Indeed, why not use this alternative single table below to represent movies?

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

Anomalies

- What's wrong with the "Wrong Movies" table?

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **Redundancy**: Unnecessary repetition of info
- **Update anomalies**: If Star Wars is 125 min, we might carelessly update row 1 but not rows 2 & 3
- **Deletion anomalies**: If Emilio Estevez is deleted from stars of Mighty Ducks, we lose all info on that movie

Functional Dependency

- **Functional dependency (A_1, …, A_n → B_1, …, B_m)**
  - If two tuples of a table R agree on attributes A_1, …, A_n, then they must also agree on attributes B_1, …, B_m
- **Example: Title, Year → Length, Film Type, Studio**
- **FD (A_1, …, A_n → B_1, …, B_m) is trivial if a B_i is an A_j**

Can you identify the FD's here?

Wrong Movies

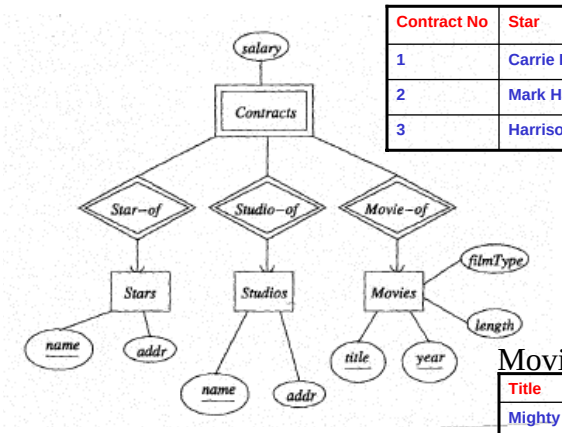
Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **Some FD's:**
  - Title, Year → Length
  - Title, Year → Film Type
  - Title, Year → Studio

Keys

- **Key**
  - A minimal set of attributes $\{A_1, \ldots, A_n\}$ that functionally determine all other attributes of a table
  - A key is trivial if it comprises the entire set of attributes of a table
- **Superkey**
  - A set of attributes that contains a key

Can you identify the keys here?



Contracts

Contract No	Star	Studio	Title	Salary
1	Carrie Fisher	Fox	Star Wars	$$$
2	Mark Hamill	Fox	Star Wars	$$$
3	Harrison Ford	Fox	Star Wars	$$$

Stars

Name	Address
Carrie Fisher	Hollywood
Mark Hamill	Brentwood
Harrison Ford	Beverly Hills

Movies

Title	Year	Length	Film Type
Mighty Ducks	1991	104	Color
Wayne's World	1992	95	Color
Star Wars	1977	124	Color

Can you identify the superkeys here?

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **Superkeys :**
  - Any set of attributes that contains {Title, Year, Star} as a subset

Boyce-Codd Normal Form

- **A relation R is in Boyce-Codd Normal Form iff whenever there is a nontrivial FD (A_1, …, A_n → B_1, …, B_m) for R, it is the case that {A_1, …, A_n} is a superkey for R**

- Theorem A1 (Codd, 1972)

  A database design has no anomalies due to FD iff all its relations are in Boyce-Codd Normal Form

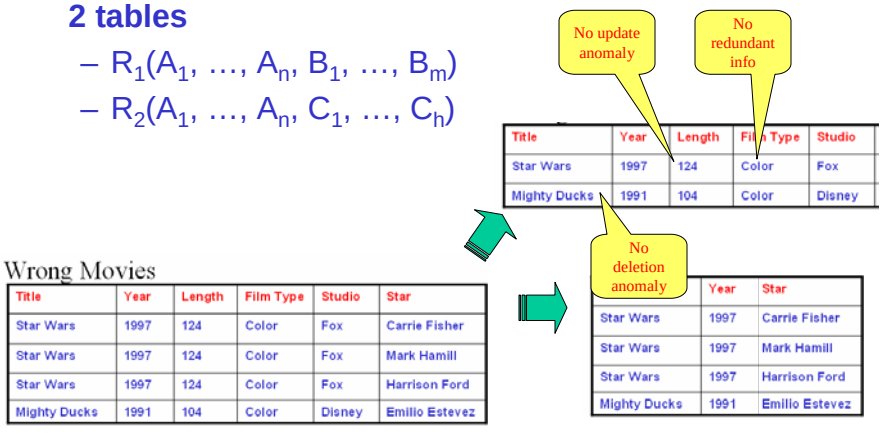
How is BCNF violated here?

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **A nontrivial FD:**
  - Title, Year → Length, Film Type, Studio
  - The LHS not superset of the key {Title,Year, Star}
  - ⇒ Violate BCNF!
- **Anomalies are due to FD's (and MVD's) whose LHS is not superkey**

Towards a Better Design

- **Use an offending FD (A_1, …, A_n → B_1, …, B_m) to decompose R(A_1, …, A_n, B_1, …, B_m, C_1, …, C_h) into 2 tables**
  - R_1(A_1, …, A_n, B_1, …, B_m)
  - R_2(A_1, …, A_n, C_1, …, C_h)



What about this table?

Star	Street	City	Title	Year
Carrie Fisher	Maple St	Hollywood	Star Wars	1977
Carrie Fisher	Locust Ln	Malibu	Star Wars	1997
Carrie Fisher	Maple St	Hollywood	Empire Strikes Back	1980
Carrie Fisher	Locust Ln	Malibu	Empire Strikes Back	1980
Carrie Fisher	Maple St	Hollywood	Return of the Jedi	1983
Carrie Fisher	Locust Ln	Malibu	Return of the Jedi	1983

- **No nontrivial FD here. So no BCNF violation**
- **Yet lots of anomalies**
- **What's happening?**
- **Addresses are independent of movie titles!**

Multivalued Dependency

- **Multivalued dependency (A_1, …, A_n →→ B_1, …, B_m)**
  - If restricted to rows of R that have fixed values of A_1, …, A_n, then the values of B_1, …, B_m are independent of attributes not among A_1, …, A_n, B_1, …, B_m

- **MDV (A_1, …, A_n →→ B_1, …, B_m) is trivial if a B_i is an A_j or {A_1, …, A_n, B_1, …, B_m} are all of R's attributes**

Can you identify the MVD's here?

Star	Street	City	Title	Year
Carrie Fisher	Maple St	Hollywood	Star Wars	1977
Carrie Fisher	Locust Ln	Malibu	Star Wars	1997
Carrie Fisher	Maple St	Hollywood	Empire Strikes Back	1980
Carrie Fisher	Locust Ln	Malibu	Empire Strikes Back	1980
Carrie Fisher	Maple St	Hollywood	Return of the Jedi	1983
Carrie Fisher	Locust Ln	Malibu	Return of the Jedi	1983

- **Some MVD's**
  - Star →→ Street, City
  - Star →→ Title, Year
- **These MVD's say**
  - A star can live in several places
  - A star can act in several movies

4th Normal Form

- **A relation R is in 4th Normal Form iff whenever there is a nontrivial MVD (A_1, …, A_n →→ B_1, …, B_m) for R, it is the case that {A_1, …, A_n} is a superkey for R**
- Theorem A2 (Fagin, 1977)

  A database design has no anomalies due to MVD iff all its relations are in 4th Normal Form
- Theorem A3 (Fagin, 1977)

  Every database design in 4th Normal Form is also in Boyce-Codd Normal Form

How is 4NF violated here?

Star	Street	City	Title	Year
Carrie Fisher	Maple St	Hollywood	Star Wars	1977
Carrie Fisher	Locust Ln	Malibu	Star Wars	1997
Carrie Fisher	Maple St	Hollywood	Empire Strikes Back	1980
Carrie Fisher	Locust Ln	Malibu	Empire Strikes Back	1980
Carrie Fisher	Maple St	Hollywood	Return of the Jedi	1983
Carrie Fisher	Locust Ln	Malibu	Return of the Jedi	1983

- **Some nontrivial MVD's**
  - Star →→ Street, City
  - Star →→ Title, Year
  - Star is not a key, so Violate 4NF
- **Anomalies are due to FD's (and MVD's) whose LHS is not superkey**

Toward a Better Design

- **Use an offending MVD (A_1, …, A_n →→ B_1, …, B_m) to decompose R(A_1, …, A_n, B_1, …, B_m, C_1, …, C_h) into 2 tables**
  - R_1(A_1, …, A_n, B_1, …, B_m)
  - R_2(A_1, …, A_n, C_1, …, C_h)

Star	Street	City	Title	Year
Carrie Fisher	Maple St	Hollywood	Star Wars	1977
Carrie Fisher	Locust Ln	Malibu	Star Wars	1997
Carrie Fisher	Maple St	Hollywood	Empire Strikes Back	1980
Carrie Fisher	Locust Ln	Malibu	Empire Strikes Back	1980
Carrie Fisher	Maple St	Hollywood	Return of the Jedi	1983
Carrie Fisher	Locust Ln	Malibu	Return of the Jedi	1983

Star	Title	Year
Carrie Fisher	Star Wars	1977
Carrie Fisher	Empire Strikes Back	1980
Carrie Fisher	Return of the Jedi	1983

Star	Street	City
Carrie Fisher	Maple St	Hollywood
Carrie Fisher	Locust Ln	Malibu

The "Invariant" Perspective

- FD's (A_1, …, A_n → B_1, …, B_m) are "invariants" of the database, as {A_1, …, A_n} determines {B_1, …, B_m}
- Paying attention such invariants leads to better database design

References

- E. F. Codd, "A relational model for large shared data banks", *CACM*, 13(6):377-387, 1970
- E. F. Codd, "Further normalization of the database relational model", in *Database Systems*, Prentice Hall, 1972
- R. Fagin, "Multivalued dependencies and a new normal form for relational databases", *ACM TODS*, 2(3):262-278, 1977

Transaction Management

Transaction

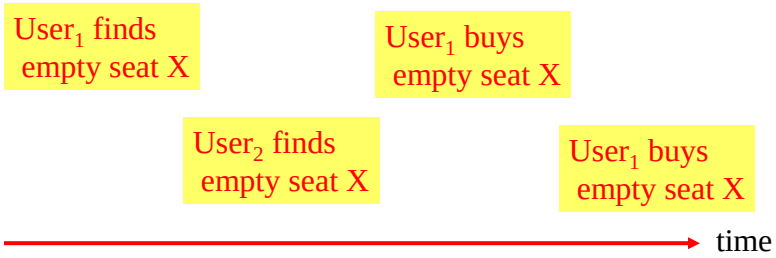
- **In real-life applications, when we need to update a db, we often need to do it in multiple steps**
- **E.g., Buying air ticket involves**
  - Find empty seat
  - Buy the ticket
- **The sequence of steps is called a transaction**

"ACID" Properties of Transactions

- **Atomicity**
  - All-or-nothing execution of a transaction
- **Consistency**
  - All transactions should preserve any consistency & integrity constraints specified on the db
- **Isolation**
  - Each transaction must appear to be executed as if no other transactions are running
- **Durability**
  - The effect of the transaction on the db must never be lost once the transaction has completed

How Things Can Go Wrong

- **Real-life db executes many transactions simultaneously**
- **If things are not carefully controlled, steps of different transactions can overlap each other dangerously …**



Serializability

- **The execution of a set of transaction is serializable if they behave as if they were run serially**
- **Hard to require a db to run serially in practice**
  - Too many transactions
  - Need parallelism to improve throughput

⇒**Need mechanism to ensure serializability**
  - Even if transactions are not executed serially, the result looks to users as if they were run serially

Schedule

- **Actions of transaction T_i**
  - $R_i(X)$: T_i reads data element X
  - $W_i(X)$: T_i writes data element X
- **Schedule S of a set of transactions Ts**
  - A sequence of actions of transactions in Ts
  - For each T_i in Ts, the actions of T_i appear in S in the same order they appear in T_i

Serial Schedule

- $R_2(A); W_2(A); R_2(B); W_2(B); R_1(A); W_1(A); R_1(B); W1(B);$
- The above is T_2 before T_1, resulting in A = 150 and B = 150
- If T_1 is run before T_2, we get A = (25 + 100) * 2 = 250 and B = (25 + 100) * 2 =250

T_1	T_2	A	B
		25	25
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	50	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		50
READ(A,t)			
t := t+100			
WRITE(A,t)		150	
READ(B,t)			
t := t+100			
WRITE(B,t)			150

Serializable Non-Serial Schedule

- $R_1(A); W_1(A); R_2(A); W_2(A); R_1(B); W_1(B); R_2(B); W_2(B)$
- So A = 250 and B = 250
- This is a serializable schedule, as it is same as running T_1 before T_2

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
READ(B,t)			
t := t+100			
WRITE(B,t)			125
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		250

Non-Serializable Schedule

- $R_1(A); W_1(A); R_2(A); W_2(A); R_2(B); W_2(B); R_1(B); W_1(B)$
- So A = 250 and B = 150
- This is diff from running T_1 before T_2 (A = B = 250) and from T_2 before T_1 (A = B = 150)
- So this schedule is non-serializable

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		50
READ(B,t)			
t := t+100			
WRITE(B,t)			150

Conflict Serializability

- A pair of consecutive actions in a schedule are in **conflict** if the behavior of at least one of the transactions involved can change if their order are interchanged
- Two schedules are **conflict equivalent** if they can be turned one into the other by a sequence of nonconflicting swaps of adjacent actions
- A schedule is **conflict serializable** if it is conflict equivalent to a serial schedule

Conflicts

- **Non-conflicts**
  - $R_i(X)$; $R_j(Y)$
    - Neither change value of any data element
  - $W_i(X)$; $R_j(Y)$, $X \neq Y$
  - $W_i(X)$; $W_j(Y)$, $X \neq Y$
- **Conflicts**
  - $R_i(X)$;$W_i(Y)$
    - Order of actions in a transaction is fixed
    - BMS
    - en by we swap the actions
  - $R_i(X)$; $W_j(X)$
    - Value of X read by Ti will be diff if we swap the actions
  - $W_i(X)$; $R_j(X)$

Any two actions of different transactions may be swapped unless
- They involve the same data element, and
- At least one is a write

Example

- **Recall this serializable schedule:**
  - $R_1(A)$; $W_1(A)$; $R_2(A)$; $W_2(A)$; $R_1(B)$; $W_1(B)$; $R_2(B)$; $W_2(B)$
  - $\Rightarrow$ $R_1(A)$; $W_1(A)$; $R_2(A)$; $R_1(B)$; $W_2(A)$; $W_1(B)$; $R_2(B)$; $W_2(B)$
  - $\Rightarrow$ $R_1(A)$; $W_1(A)$; $R_1(B)$; $R_2(A)$; $W_2(A)$; $W_1(B)$; $R_2(B)$; $W_2(B)$
  - $\Rightarrow$ $R_1(A)$; $W_1(A)$; $R_1(B)$; $R_2(A)$; $W_1(B)$; $W_2(A)$; $R_2(B)$; $W_2(B)$
  - $\Rightarrow$ $R_1(A)$; $W_1(A)$; $R_1(B)$; $W_1(B)$; $R_2(A)$; $W_2(A)$; $R_2(B)$; $W_2(B)$
- **Same as running T_1 before T_2**
- **Thus it is conflict serializable**

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
READ(B,t)			
t := t+100			
WRITE(B,t)			125
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		250

Conflict Serializability vs Serializability

- Proposition B1

  Every conflict serializable schedule is serializable, but some serializable schedule is not conflict serializable

S_1: $W_1(Y)$; $W_1(X)$; $W_2(Y)$; $W_2(X)$; $W_3(X)$
S_2: $W_1(Y)$; $W_2(Y)$; $W_2(X)$; $W_1(X)$; $W_3(X)$

- **S_1 is a serial schedule**
- **S_2 is serializable (X and Y have same final value as S_1) but not conflict serializable**

Enforcing Serializability by Locks

- **A collection of transactions performing their actions in an uncontrolled manner is unlikely to result in a serializable schedule**

- **Need a mechanism to prevent orders of actions that lead to unserializable schedule…**

⇒ **Locks**

A transaction obtains locks on data elements it accesses to prevent other transactions from accessing these elements at the same time, thus avoiding the risk of unserializability

Locks

- **$L_i(X)$: T_i requests a lock on data element X**
- **$U_i(X)$: T_i releases its lock on data element X**

- **Must satisfy**
  - Consistency of transactions: Whenever T_i has an action $R_i(X)$ or $W_i(X)$, there is a previous action $L_i(X)$ with no intervening $U_i(X)$, and there is a subsequent $U_i(X)$
  - Legality of schedules: Whenever there are actions $L_i(X)$ followed by $L_j(X)$, there is a $U_i(X)$ somewhere between these two actions

Unfortunately …

- **A legal schedule of consistent transactions is not necessarily serializable**
- **Example**

T_1	T_2	A	B
		25	25
$l_1(A)$; $r_1(A)$;			
A := A+100;			
$w_1(A)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;		
	A := A*2;		
	$w_2(A)$; $u_2(A)$;	250	
	$l_2(B)$; $r_2(B)$;		
	B := B*2;		
	$w_2(B)$; $u_2(B)$;		50
$l_1(B)$; $r_1(B)$;			
B := B+100;			
$w_1(B)$; $u_1(B)$;			150

Recall serial execution of T_1 and T_2 gives A = B = 250 or A = B = 150

T_2 changes value of B, affecting what T_1 sees

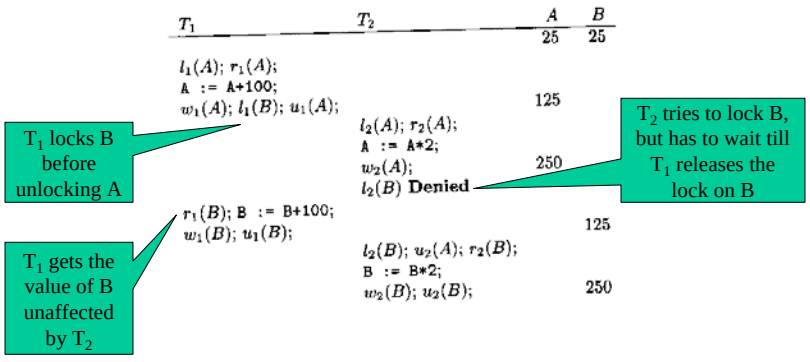
Guaranteeing Serializability

- **Two-Phase Locking (2PL): In every transaction, all lock requests should precede all unlock requests**
- Theorem B2

  A conflict-equiv serial schedule for a schedule of 2PL transactions is the one in which transactions are in the same order as their first unlocks
- Corollary B3

  2PL guarantees that a legal schedule of consistent transactions is conflict serializable

Example



The "Invariant" Perspective

- **"A conflict-equiv serial schedule for a schedule of 2PL transactions is the one in which transactions are in the same order as their first unlocks" (Theorem B2) is an invariant property of 2PL**
- **"Consistency of transactions", "legality of schedules", "conflict serializability" are all invariants maintained by DBMS via 2PL to guarantee ACID properties**

References

- **K. P. Eswaran et al, "The Notions of Consistency and Predicate Locks in a Database System", *CACM*, 19(11):624-633, 1976**
- **H. T. Kung, J. T. Robinson, "Optimistic Concurrency Control", *ACM TODS*, 6(2):312-326, 1981**
- **P. A. Bernstein, N. Goodman, "Timestamp-Based Algorithms for Concurrency Control in Distributed Database Systems", *Proc. VLDB 1980*, pp 285-300**
- **H. F. Korth, "Locking Primitives in a Database System", *JACM*, 30(1):55-79, 1983**

Query Languages

Relational Algebra

$$\frac{R:\{u\}, S:\{u\}}{R \cup S:\{u\}} \qquad \frac{R:\{u\}, S:\{u\}}{R - S:\{u\}} \qquad \frac{R:\{u\}, S:\{v\}}{R \otimes S:\{u \times v\}}$$

$$\frac{R:\{u_1 \times u_2\}}{\pi_i R:\{u_i\}} \qquad \frac{R:\{u\},\ f:u \rightarrow bool}{\sigma_f R:\{u\}}$$

Title	Year	Length	Film Type
Mighty Ducks	1991	104	Color
Wayne's World	1992	95	Color
Star Wars	1977	124	Color

$\pi_{\text{Title, Year}}$

Title	Year
Mighty Ducks	1991
Wayne's World	1992
Star Wars	1977

$\sigma_{\text{Year} = 1997}$

Title	Year	Length	Film Type
Star Wars	1977	124	Color

Closure & Composition

- Proposition C1
  Operators of RA is closed over relations
- Corollary C2
  Operators of RA can be composed to form complex queries
- **Example: $R\ |x|_f\ S := \sigma_f (R \otimes S)$**

A More Convenient Syntax

Select x.L from R as x where C(x)

- **$R \otimes S$:= select x.*, y.* from R x, S y**
- **π_L R := select x.L from R x**
- **σ_f R := select x.* from R x where f(x)**
- **$R\ |x|_f\ S$:= select x.*, y.* from R x, S y where f(x,y)**
- **$R \cup S$:= select x.* from R x union select y.* from S y**
- **R – S := select x.* from R x except select y.* from S y**

Some Laws for Query Optimization

- $\sigma_f(\sigma_g(S)) = \sigma_{f \text{ AND } g} S$
- $\sigma_{f \text{ OR } g} S = \sigma_f S \cup \sigma_g S$
- $\sigma_f(R \cup S) = \sigma_f S \cup \sigma_f S$
- $\sigma_f(R - S) = \sigma_f(R) - S$
- $\sigma_f(R \,|x|_g\, S) = \sigma_f(R) \,|x|_g\, S$, if R has all attributes mentioned in f
- $\sigma_f(R \,|x|_g\, S) = R \,|x|_g\, \sigma_f(S)$, if S has all attributes mentioned in f

The RHS is usually more efficient than the LHS

The "Invariant" Perspective

- The "laws" for query optimization are "invariants" of RA
- Paying attention to them leads to more efficient queries

References

- E. F. Codd, "A Relational Model for Large Shared Data Banks", *CACM*, 13(6):377-387, 1970
- M. M. Astrahan et al, "System R: A Relational Approach to Data Management", *ACM TODS*, 1(2):97-137, 1976
- E. Wong, K. Youssefi, "Decomposition---A Strategy for Query Processing", *ACM TODS*, 1(3):223-241, 1976
- P. Griffiths-Selinger et al, "Access Path Selection in a Relational Database System", *Proc SIGMOD 1979*, pp 23-34
- G. Graefe, "Special Issue on Query Processing in Commercial DBMS", *Data Engineering*, 16(4), 1993
- S. Chaudhuri, "An Overview of Query Optimization in Relational Systems", *Proc PODS 1998*, pp 34-43

A Twist in the Tale: Limit of Expressive Power of Relational Algebra

Complexity of Typical Queries

- **A typical query**

select $x_1.*$, …, $x_n.*$
from R_1 x_1, …, R_n x_n
where $x_1.l_1 = x_2.l_1$ and $x_2.l_2 = x_3.l_2$ and … and $x_{n-1}.l_{n-1} = x_n.l_{n-1}$

- **Expected complexity: $O(N_1 * \log N_2 * \ldots * \log N_n)$**

Polynomiality of Relational Algebra

$$\frac{R:\{u\}, S:\{u\}}{R-S:\{u\}} \qquad \frac{R:\{u\}, S:\{u\}}{R\cup S:\{u\}} \qquad \frac{R:\{u\}, S:\{v\}}{R\otimes S:\{u\times v\}}$$

$$\frac{R:\{u_1\times u_2\}}{\pi_i R:\{u_i\}} \qquad \frac{R:\{u\},\ f:u\to bool}{\sigma_f R:\{u\}}$$

- Proposition D1
  All queries definable in RA are in PTIME

- **Is there a PTIME query that is RA inexpressible?**

Transitive Closure

```
Grandfather(R) :=
        select x.dad, y.son
        from R x, R y
        where x.son = y.dad
```

```
Greatgrandfather(R) :=
        select x.dad, z.son
        from R x, R y, R z
        where x.son = y.dad
        and y.son = z.dad
```

- **It is possible to define "great grand father to the n^{th} generation" for any arbitrary "n" in RA**
- **But it is possible to define an "ancestor-descendent" table in RA?**
- **If it is not possible, how do you prove this? (There are zillions of programs that you can write using RA. You can't possible check them one by one!)**

Degrees of a Graph

- **Think of the "father-son" table as a graph**
  - Each person is a node
  - There is an edge from a father to each of his sons
- **The "degree" of a node is the number of incoming and outgoing edges**
- **The degrees of a graph is the set of degrees of its nodes**

Number of Distinct Degrees

- **Suppose R is such that each father has up to 2 children**
- **Grandfather(R) is a graph with 4 distinct degrees**
  - 1: both x and y have 1 son, or x has 2 sons and one son has one son and one son has no son
  - 2,
  - 3,
  - 4
- **Given max number of children a father can have, can predict no. of distinct degrees in Grandfather(R), Greatgrandfather(R), …**

Number of Distinct Degrees

```
MessUp(R) :=
        select x.dad, y.son
        from R x, R y
```

- **Can't predict what are the degrees in MessUp(R)**
- **Can predict that MessUp(R) has at most 2 distinct degrees regardless of what R is**

Bounded Degree Property

A language $\mathcal{L}$ has bounded degree property if

for every function f, on graphs, definable in $\mathcal{L}$, and for any number k,

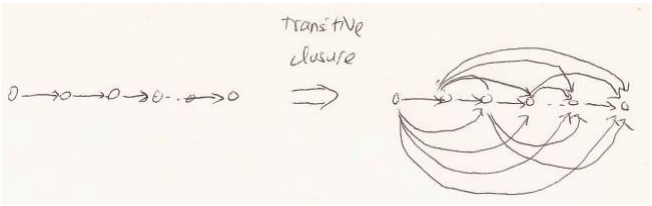
there is a number c such that
for any graph $\mathcal{G}$ with $\deg(\mathcal{G}) \subseteq \{0, 1, \ldots, k\}$,
it is the case that $c \geq \text{card}(\deg(f(\mathcal{G})))$

That is, $\mathcal{L}$ cannot define a function that produces complex graphs from simple graphs

A Deep Invariant of RA Queries

- Theorem D2
  RA has the bounded degree property

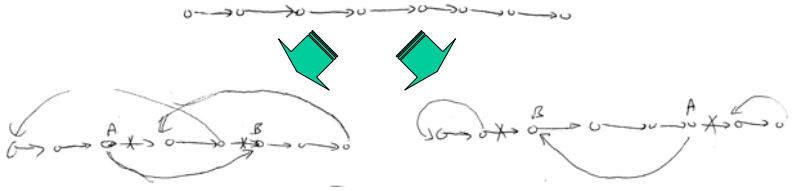
- Corollary D3
  RA cannot define transitive closure. In fact, it cannot even define transitive closure restricted to chains



More Consequences that are hard to prove otherwise …

- Proposition D4

  RA can test whether a graph is a single cycle iff RA can express transitive closure of a chain



- Corollary D5

  RA cannot test whether a graph is a single cycle

The "Invariant" Perspective

- **The bounded degree property is an "invariant" of RA**
- **It points out the deep structure of all RA queries**
- **Exploiting it, we can readily solve many ad hoc problems that are hard to solve otherwise**

References

- L. Wong, "Normal Forms and Conservative Extension Properties for Query Languages over Collection Types", *JCSS*, 52(3):495--505, 1996.
- L. Libkin, L. Wong, "Query Languages for Bags and Aggregate Functions", *JCSS*, 55(2):241--272, 1997.
- G. Dong, L. Libkin, L. Wong, "Local Properties of Query Languages", *TCS*, 239:277--308, 2000.

What have we learned?

- An invariant may reflect the deep structure of the data ⇒ better database design
- An invariant may maintain serializability of concurrent transactions ⇒ higher throughput
- An invariant may be a "law" for query optimization ⇒ more efficient queries
- An invariant may reflect the deep structure of a query language ⇒ easy solution to ad hoc problems that are hard to solve otherwise

If we still have time, …

Beyond Relational Algebra

Nested Relational Calculus (NRC)

The complex object types are:

$$s, t ::= \mid bool \mid b \mid s \times t \mid \{s\}$$

The expression constructs are:

$$\frac{}{x^s : s} \qquad \frac{e_1 : s \quad e_2 : t}{(e_1, e_2) : s \times t} \qquad \frac{e : s \times t}{\pi_1\ e : s \quad \pi_2\ e : t}$$

$$\frac{}{true\ : bool} \qquad \frac{}{false\ : bool} \qquad \frac{e_1 : bool \quad e_2 : s \quad e_3 : s}{if\ e_1\ then\ e_2\ else\ e_3 : s}$$

$$\frac{}{\{\}^s : \{s\}} \qquad \frac{e : s}{\{e\} : \{s\}} \qquad \frac{e_1 : \{s\} \quad e_2 : \{s\}}{e_1 \bigcup e_2 : \{s\}}$$

$$\frac{e_1 : \{s\} \quad e_2 : \{t\}}{\cup\{e_1 \mid x^t \in e_2\} : \{s\}} \qquad \frac{e : \{s\}}{empty\ e : bool} \qquad \frac{e_1 : s \quad e_2 : s}{e_1 = e_2 : bool}$$

Explanation

- π_1 *e* **stands for the first component of the pair *e***
  Eg: π_1 (o_1,o_2) = o_1
- $\cup\{e_1 \mid x \in e_2\}$ **stands for the set obtained by combining the results of applying the function *f*(*x*) = e_1 to each element of e_2**
  Eg: $\cup\{\{x, x+1\} \mid x \in \{1,2,3\}\} = \{1,2,3,4\}$

Examples

- **Relational projection**
  $\Pi_2(R) := \cup\{\{\pi_2\ x\} \mid x \in R\}$
- **Relational selection**
  $\sigma(p)(R) := \cup\{\text{if } p(x) \text{ then } \{x\} \text{ else } \{\} \mid x \in R\}$
- **Cartesian product**
  $\otimes(R,S) := \cup\{\cup\{\{(x,y)\} \mid x \in R\} \mid y \in S\}$

Conservative Extension Property

A language $\mathcal{L}$ has conservative extension property if

for every function f definable in $\mathcal{L}$,
there is an implementation of f in $\mathcal{L}$ such that

for any input i and corresponding output o,

each intermediate data item created in the course of executing f on i to produce o has nesting complexity less than that of i and o

Expressive Power of NRC

- Proposition E1 (Tannen, Buneman, Wong, ICDT92)

  NRC has the same expressive power as Schek&Scholl, Thomas&Fischer, etc.

- Theorem E2 (Wong, PODS93)

  NRC has the conservative extension property at all input/output types

- Corollary E3

  Every function from flat relations to flat relations expressible in NRC is expressible in RA

Theoretical Reconstruction of SQL

Expressions of $\mathcal{NRC}(\mathbb{Q}, +, \cdot, -, \div, \Sigma, =, \leq^{\mathbb{Q}})$ are those of $\mathcal{NRC}$ plus the following

$$\frac{e_1 : \mathbb{Q} \quad e_2 : \mathbb{Q}}{e_1 + e_2 : \mathbb{Q}} \qquad \frac{e_1 : \mathbb{Q} \quad e_2 : \mathbb{Q}}{e_1 \cdot e_2 : \mathbb{Q}} \qquad \frac{e_1 : \mathbb{Q} \quad e_2 : \mathbb{Q}}{e_1 \div e_2 : \mathbb{Q}}$$

$$\frac{e_1 : \mathbb{Q} \quad e_2 : \mathbb{Q}}{e_1 - e_2 : \mathbb{Q}} \qquad \frac{e_1 : \mathbb{Q} \quad e_2 : \{s\}}{\Sigma\{\!|e_1 \mid x^s \in e_2|\!\} : \mathbb{Q}} \qquad \frac{e_1 : \mathbb{Q} \quad e_2 : \mathbb{Q}}{e_1 \leq e_2 : bool}$$

Semantics. $\Sigma\{\!|e_1 \mid x \in e_2|\!\} = f(o_1) + \ldots + f(o_n)$, where f is the function $f(x) = e_1$ $\{o_1, \ldots, o_n\}$ is the set e_2.

Example Aggregate Functions

- **Count the number of records**
  count(*R*) := Σ{| 1 | *x* ∈ *R* |}
- **Total the first column**
  total$_1$(*R*) := Σ{| π_1 *x* | *x* ∈ *R* |}
- **Average of the first column**
  ave$_1$(*R*) := total$_1$(*R*) ÷ count(*R*)
- **A totally generic query expressible in SQL but inexpressible in RA**
  eqcard(*R*,*S*) := count(*R*) = count(*S*)

Expressive Power of NRC(Q,+,•,–,÷,Σ,=, $\geq^Q$)

- Proposition (Libkin, Wong, DBPL93)

  NRC(Q,+,•,–,÷,Σ,=, $\geq^Q$) captures "standard" SQL

- Theorem E4 (Libkin, Wong, PODS94)

  NRC(Q,+,•,–,÷,Σ,=, $\geq^Q$) has the conservative extension property at all input/output types

- Corollary E5

  Every function from flat relations to flat relations is expressible in NRC(Q,+,•,–,÷,Σ,=, $\geq^Q$) iff it is also expressible in SQL

Bounded Degree Property

A language $\mathcal{L}$ has bounded degree property if

for every function f, on graphs, definable in $\mathcal{L}$, and for any number k,

there is a number c such that

for any graph $\mathcal{G}$ with deg($\mathcal{G}$) $\subseteq$ { 0, 1, …, k},
it is the case that $c \geq$ card(deg(f($\mathcal{G}$)))

That is, $\mathcal{L}$ cannot define a function that produces complex graphs from simple graphs

Expressive Power of NRC(Q,+,•,–,÷,Σ,=, $\geq^Q$)

- Theorem E6 (Dong, Libkin, Wong, ICDT97)

  NRC(Q,+,•,–,÷,Σ,=, $\geq^Q$) has the bounded degree property

- Corollary E7
  - Transitive closure of unordered graphs cannot be expressed in SQL
  - Parity test on cardinality of unordered graphs cannot be expressed in SQL
  - Transitive closure of linear chains cannot be expressed in SQL
  - ...