CIS 560 – Database System Concepts

Lecture 12

Transactions in SQL and Relational Algebra

September 23, 2013

Credits for slides: Suciu, Chang.

Outline

Last time:

- DB Design: Normalization (3.3-3.4)

Today:

- Transactions in SQL (6.6)
- Relational algebra (Sections 2.4 and 5.1-5.2)

Next:

- Relational algebra (Sections 2.4 and 5.1-5.2)
- Introduction to Database Programming (Ch. 9)
  - Connect to DB and call SQL from Java

2

Review

- BCNF
- 3NF
- Lossless decomposition
- Preserving functional dependencies

3

Transactions

- The problem: An application must perform *several* writes and reads to the database, as a unit.
  - Example: Two people attempt to book the last seat on a flight.
- Solution: multiple actions of the application are bundled into one unit called *Transaction.*
  - Transactions guarantee certain properties to hold that prevent problems.

4

The World Without Transactions

- Just write applications that talk to databases
- Rely on operating systems for scheduling, and for concurrency control
- What can go wrong?
  - Three famous anomalies
  - Other anomalies are possible (but not so famous)

5

Lost Updates

Client 1:	Client 2:
UPDATE Customer SET rentals= rentals + 1 WHERE cname= ‘Fred’	UPDATE Customer SET rentals= rentals + 1 WHERE cname= ‘Fred’

Two people attempt to rent two movies for Fred, from two different terminals. What happens?

6

Inconsistent Read

```
Client 1: rent-a-movie
x = SELECT rentals FROM Cust
      WHERE cname= 'Fred'


if (x < 5)
  { UPDATE Cust
    SET rentals= rentals + 1
    WHERE cname= 'Fred' }
else println("Denied !")
```

```
Client 2: rent-a-movie
x = SELECT rentals FROM Cust
      WHERE cname= 'Fred'


if (x < 5)
  { UPDATE Cust
    SET rentals= rentals + 1
    WHERE cname= 'Fred' }
else println("Denied !")
```

What's wrong ?

7

Inconsistent Read

```
Client 1: rent-two-movies
x = SELECT rentals FROM Cust
      WHERE cname= 'Fred'

if (x < 4) { /* movie 1…*/
    UPDATE Cust
    SET rentals= rentals + 1
    WHERE cname= 'Fred'

    /* …and movie 2…*/
    UPDATE Cust
    SET rentals= rentals + 1
    WHERE cname= 'Fred'
}
else println("Denied !")
```

```
Client 2: rent-a-movie
x = SELECT rentals FROM Cust
      WHERE cname= 'Fred'


if (x < 5)
  { UPDATE Cust
    SET rentals= rentals + 1
    WHERE cname= 'Fred' }
else println("Denied !")
```

What's wrong ?

8

Inconsistent Read

Client 1: move from gizmo→gadget

```
UPDATE Products
SET quantity = quantity + 5
WHERE product = 'gizmo'

UPDATE Products
SET quantity = quantity - 5
WHERE product = 'gadget'
```

Client 2: inventory….

```
SELECT sum(quantity)
FROM Product
```

What's wrong ?

9

Dirty Reads

Client 1: transfer \$100 acc1→ acc2

```
X = Account1.balance
Account2.balance += 100

If (X>=100) Account1.balance -=100
else { /* rollback ! */
        account2.balance -= 100
        println("Denied !")
```

Client 2: transfer \$100 acc2 → acc3

```
Y = Account2.balance
Account3.balance += 100

If (Y>=100) Account2.balance -=100
else { /* rollback ! */
        account3.balance -= 100
        println("Denied !")
```

What's wrong ?

The Three Famous Anomalies

- Lost update
  - Two tasks T and T' both modify the same data
  - T and T' both commit
  - Final state shows effects of only T, but not of T'
- Inconsistent read:
  - One task T might see some but not all changes made by T'
- Dirty read
  - T reads data written by T' while T' has not committed
  - What can go wrong: T' writes more data (which T has already read), or T' aborts

11

Protection against crashes

Client 1:

```
UPDATE Accounts
SET balance= balance - 500
WHERE name= 'Fred'

UPDATE Accounts
SET balance = balance + 500
WHERE name= 'Joe'
```

Crash !

What's wrong ?

12

Transactions

Ensure two things:

- Concurrency control
  - Making sure simultaneous transactions don't interfere with one another
- Recovery
  - Taking action to restore the DB to a consistent state

13

Definition

- **A transaction** = one or more operations, which reflects a single real-world transition
  - Happens completely or not at all
- Examples
  - Transfer money between accounts
  - Rent a movie; return a rented movie
  - Purchase a group of products
  - Register for a class (either waitlisted or allocated)
- By using transactions, all previous problems disappear

14

Transactions in Applications

- Default: each statement = one transaction
- Multi-statement transactions:

START TRANSACTION
[SQL statements]
COMMIT or ROLLBACK (=ABORT)

15

Revised Code

Client 1: rent-a-movie

```
START TRANSACTION
x = SELECT rentals FROM Cust
      WHERE cname= 'Fred'

if (x < 5)
  { UPDATE Cust
    SET rentals= rentals + 1
    WHERE cname= 'Fred' }
else println("Denied !")
COMMIT
```

Client 2: rent-a-movie

```
START TRANSACTION
x = SELECT rentals FROM Cust
      WHERE cname= 'Fred'

if (x < 5)
  { UPDATE Cust
    SET rentals= rentals + 1
    WHERE cname= 'Fred' }
else println("Denied !")
COMMIT
```

16

Revised Code

Client 1: transfer \$100 acc1→ acc2
START TRANSACTION
X = Account1.balance; Account2.balance += 100

If (X>=100) { Account1.balance -=100; COMMIT }
else {println("Denied !"; ROLLBACK)

Client 2: transfer \$100 acc2→ acc3
START TRANSACTION
X = Account2.balance; Account3.balance += 100

If (X>=100) { Account2.balance -=100; COMMIT }
else {println("Denied !"; ROLLBACK)

17

Using Transactions

Very easy to use:

- START TRANSACTION
- COMMIT
- ROLLBACK

But what EXACTLY do they mean ?

- Popular culture: ACID
- Underlying theory: serializability

18

Transaction Properties ACID

- Atomic
  - State shows either all the effects of a transaction, or none of them
- Consistent
  - A transaction moves from a state where integrity holds, to another where integrity holds
- Isolated
  - Effect of transactions is the same as transactions running one after another (i.e., looks like batch mode)
- Durable
  - Once a transaction has committed, its effects remain in the database

19

ACID: Atomicity

- Two possible outcomes for a transaction
  - It *commits*: all the changes are made
  - It *aborts*: no changes are made
- That is, transaction's activities are all or nothing

20

ACID: Isolation

- A transaction executes concurrently with other transaction
- Isolation: the effect is as if each transaction executes in isolation of the others

21

ACID: Consistency

- The database satisfies integrity constraints
  - Account number is unique
  - Stock amount can't be negative
  - Sum of *debits* and of *credits* is 0
- Constraints may be explicit or implicit
- How consistency is achieved:
  - Applications preserve consistency, assuming they run atomically, and they run in isolation
  - The system ensures atomicity and isolation

22

ACID: Durability

- The effect of a transaction must continue to exists after the transaction, or the whole program has terminated
- Means: write data to disk
- Sometimes also means recovery

23

ROLLBACK

- If the application gets to a place where it can't complete the transaction successfully, it can execute ROLLBACK
- This causes the system to "abort" the transaction
  - The database returns to the state without any of the previous changes made by activity of the transaction

24

Reasons for Rollback

- Users change their minds (“ctl-C”/cancel)
- Explicit in program, when application program finds a problem
  - E.g. when the # of rented movies > max # allowed
  - Use it freely in your next programming assignment!
- System-initiated abort
  - System crash
  - Housekeeping, e.g. due to timeouts

25

Relational Algebra

26

The WHAT and the HOW

- In SQL we write WHAT we want to get form the data
- The database system needs to figure out HOW to get the data we want
- The passage from WHAT to HOW goes through the Relational Algebra

27

SQL = WHAT

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

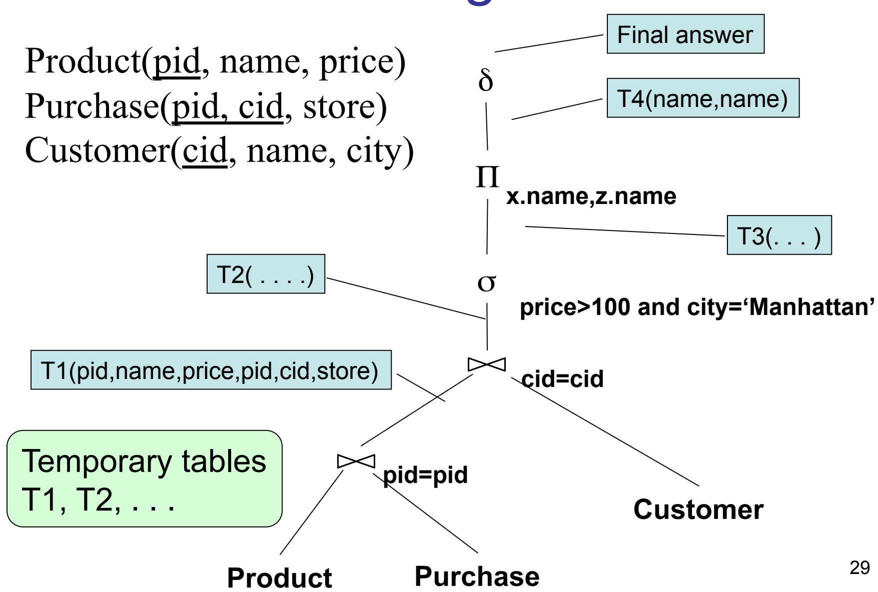
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
      x.price > 100 and z.city = 'Manhattan'
```

It's clear WHAT we want, unclear HOW to get it

28

Relational Algebra = HOW

Product(<u>pid</u>, name, price)
Purchase(<u>pid, cid</u>, store)
Customer(<u>cid</u>, name, city)



29

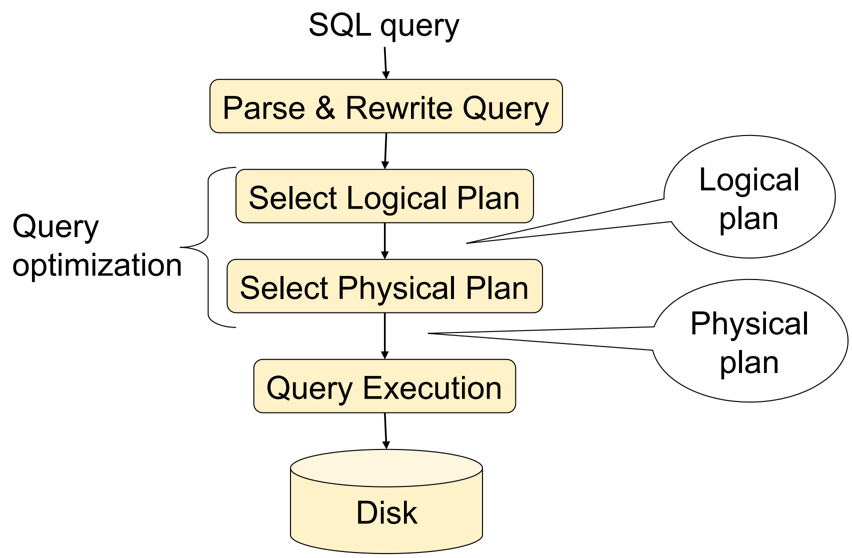
Relational Algebra = HOW

The order is now clearly specified:

Iterate over PRODUCT…
…join with PURCHASE…
…join with CUSTOMER…
…select tuples with Price>100 and City='Manhattan'…
…eliminate duplicates…
…and that's the final answer !

30

Steps of the Query Processor



Query Plans

- **Logical query plan**: an extended relational algebra tree
- **Physical query plan**: with additional annotations at each node
  - Access method to use for each relation
  - Implementation to use for each relational operator

Relations

- A relation is a set of tuples
  - Sets: {a,b,c}, {a,d,e,f}, { }, . . .
- But, commercial DBMSs implement relations that are bags rather than sets
  - Bags: {a, a, b, c}, {b, b, b, b, b}, . . .

33

Sets versus Bags

- Relational Algebra has two flavors:
  - Over sets: theoretically elegant but limited
  - Over bags: needed for SQL queries + more efficient
    - Example: Compute average price of all products
- We discuss set semantics
  - We mention bag semantics only where needed

34

Relational Algebra

- Query language associated with the relational model
- Queries specified in an operational manner
  - A query gives a step-by-step procedure
- Relational operators
  - Take one or two relation instances as argument
  - Return one relation instance as result
  - Easy to compose into relational algebra expressions

35

Relational Algebra (1/3)

The Basic Five operators:

- Union: $\cup$
- Set difference: -
- Selection: $\sigma_{condition}(S)$
  - Condition is a Boolean combination ($\wedge, \vee$) of terms
  - Term is: `attr op const, attr op attr`
  - Op is: <, <=, =, !=, >=, or >
- Projection: $\pi_{list\text{-}of\text{-}attributes}(S)$
- Cross-product or Cartesian product: $\times$

36

Relational Algebra (2/3)

Derived or auxiliary operators:

- Intersection ($\cap$)
- Join $R \bowtie_\theta S = \sigma_\theta(R \times S)$
- Variations of joins
  - natural, equijoin, theta join
  - outer-join and semi-join
- Rename $\rho_{B1,\ldots,Bn}(S)$

37

Relational Algebra (3/3)

Extensions for bags:

- Duplicate elimination: δ
- Group by: γ [Same symbol as aggregation]
  - Partitions tuples of a relation into "groups"
- Sorting: τ

Other extensions:

- Aggregation: γ (min, max, sum, average, count)

38

Union and Difference

R1 ∪ R2
Example: ActiveEmployees ∪ RetiredEmployees

R1 – R2
Example: AllEmployees – RetiredEmployees

Be careful when applying to bags!

39

What about Intersection?

- Derived operator using minus

R1 ∩ R2 = R1 – (R1 – R2)

Example: UnionizedEmployees ∩ RetiredEmployees

40

Selection

- Returns all tuples which satisfy a condition

$$\sigma_{condition}(R)$$

- Examples
  - $\sigma_{Salary > 40000}$ (Employee)
  - $\sigma_{name = \text{“Smith”}}$ (Employee)

Maps to the WHERE clause in SQL

41

Employee

SSN	Name	Salary
1234545	John	200000
5423341	Smith	600000
4352342	Fred	500000

$\sigma_{Salary > 40000}$ (Employee)

SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

42

Projection

- Eliminates columns

$$\Pi_{A1,\ldots,An}(R)$$

- Example: project on Name and Salary:
  - $\Pi_{\text{Name, Salary}}$ (Employee)
  - Answer(Name, Salary)

Semantics differs over set or over bags

43

Employee

SSN	Name	Salary
1234545	John	200000
5423341	John	600000
4352342	John	200000

$\Pi_{\text{Name,Salary}}$ (Employee)

Name	Salary
John	20000
John	60000
John	20000

Bag semantics

Name	Salary
John	20000
John	60000

Set semantics

Which is more efficient to implement?

44

Selection & Projection Examples

Patient

no	name	zip	disease
1	p1	98125	flu
2	p2	98125	heart
3	p3	98120	lung
4	p4	98120	heart

$\pi_{zip,disease}$(Patient)

zip	disease
98125	flu
98125	heart
98120	lung
98120	heart

$\sigma_{disease=\text{'heart'}}$(Patient)

no	name	zip	disease
2	p2	98125	heart
4	p4	98120	heart

π_{zip} ($\sigma_{disease=\text{'heart'}}$(Patient))

zip
98120
98125

Cartesian Product

- Each tuple in R1 with each tuple in R2

$$\boxed{R1 \times R2}$$

- Very rare in practice; mainly used to express joins

46

Employee

Name	SSN
John	999999999
Tony	777777777

Dependent

EmpSSN	DepName
999999999	Emily
777777777	Joe

Employee $\times$ Dependent

Name	SSN	EmpSSN	DepName
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

47

Renaming

- Changes the schema, not the instance

$$\boxed{\rho_{B1,\ldots,Bn}(R)}$$

- Example:
  - $\rho_{\text{FirstName, SocSecNo}}$(Employee)
    → Employee(FirstName, SocSecNo)

48

Renaming

Employee

Name	SSN
John	999999999
Tony	777777777

$\rho_{\text{FirstName, SocSecNo}}(\text{Employee})$

Employee

FirstName	SocSecNo
John	999999999
Tony	777777777

49