CS 348: Intro to Database Management

Michael Noukhovitch

Winter 2015, University of Waterloo

Notes written from Grant Weddel's lectures.

Contents

1 Introduction

1.1 DBMS

1.1.1 Definitions

Database: a large and persistent collection of data
DBMS: a program that manages details for storage and access to a db
to abstract common functions and create a uniform interface we need:

- **data model**: all data stored uniformally
- **access control**: authorization to modify/view
- **concurrency control**: multiple applications can access at same time
- **database recovery**: nothing is lost
- **database maintenance**

1.1.2 Three-Level Schema

schema: a description of the data interface to the database
external schema: what the app and user see
conceptual schema: description of the logical structure of the data
physical schema: description of physical aspects (storage algorithms . . .)

DBMS allows the data to be stored via the physical schema, reasoned via the conceptual schema, and accessed via the external schema.

1.1.3 Interfacing

Interfacing to DBMS, we can interact with it through:
Data Definition Language: specifies schemas

- may be different for each schema
- the **data dictionary** (or **catalog**) stores the information

Data Manipulation Language: specifies queries and updates *(e.g SQL)*

- navigational (procedural)
- non-navigational (declarative)

1.2 Big Ideas

There are three big ideas which have influenced the creation and development of databases

1.2.1 Data Independence

data independence allows each schema to be independant of the others

- **physical independance**: application immune to changes in storage structure
- **logical independence**: application immune to changes in data organization

1.2.2 Transaction

Transaction: an application-specified atomic and durable unit of work
ACID: transaction properties ensured by the DBMS

- **atomic**: a transaction cannot be split up
- **consistency**: each transaction preserves consistency
- **isolated**: concurrent transaction don't interfere with each other
- **durable**: once completed, changes are permanent

2 Relational Model

2.1 Definitions

Relational model: all information is organized in (flat) relations

- powerful and declarative query language
- semantic integrity constraints (using first order logic)
- data independence

2.2 Properties

- based on finite set theory
  - attribute ordering *not strictly necessary*
  - tuples identified by attribute values
  - instance has set semantics *no ordering, no duplicates*
- all attribute values are atomic
- **degree**: number of attributes in schema
- **cardinality**: number of tuples in instance

We can algebraically define databases as a finite set of relation schemas

2.3 Relations vs SQL Tables

SQL has extensions on top of the relational model:

1. semantics of instances:
   - relations are **sets** of tuples
   - tables are **multisets** (bags) of tuples
2. unknown values: SQL includes 'null'

3 Relation Algebra

3.1 Primary Operators

- **Relation Name**: R
- **Selection**: $\sigma_{condition}(E)$ satisfies some condition
- **Projection**: $\pi_{attributes}(E)$ only includes these attributes
- **Rename**: $\rho(R(\bar{F}), E)$
  - $\bar{F}$ is a list of $oldname \mapsto newname$
- **Product**: $E_1 x E_2$

3.2 Joins

- **Conditional Join**: $E_1 \bowtie_{condition} E_2$
- **Natural Join**: $E_1 \bowtie E_2$ common attributes

3.3 Set Operators

Schemas R and S must be **union compatible**: have same number (and type) of fields

- **Union**: $R \cup S$
- **Difference**: $R - S$
- **Intersection**: $R \cap S$

4 SQL

4.1 SQL Standard

Data Manipulation Language : query and modify tables

Data Definition Language : create tables and enforce access/security

Example 4.1. Basic query block

```
select attribute-list
from relation-list
[where condition]
```

4.2 DML

4.2.1 Null

A necessary evil that indicates unknown or missing data

- test using `is (not)NULL`
- expressions with `NULL` e.g. `x + NULL = NULL`
- `where` treats `NULL` like `False`

4.2.2 Subquery

`where` supports predicates as part of its clause

Example 4.2. select all employees with the highest salary

```
select empno, lastname
from employee
where salary >= all
    ( select salary
     from employee )
```

4.2.3 Ordering

No ordering can be assumed unless you use `order by`

4.2.4 Grouping

`group by` allows you to aggregate results

Example 4.3. for each dept, list number of employees and combined salary

```
select deptno, deptname, sum(salary) as totalsalary,
    count(*)as employees
from department d, employee e
where e.workdept = d.deptno
group by deptno, deptname
```

`having` is like `where` for groups

Example 4.4. list average salary for each dept >= 4 people

```
select deptno, deptname, avg(salary) as MeanSalary
    count(*)as employees
from department d, employee e
where e.workdept = d.deptno
group by deptno, deptname
having count(*) >= 4
```

4.3 DDL

4.3.1 Table

create : creates a table

alter : change the table

drop : delete the table

Example 4.5. create table

```
create table Employee (
EmpNo char(6),
FirstName varchar(12),
HireDate date
)
```

4.3.2 Data Types

- integer
- decimal(p,q)
- float(p)
- char(n)
- varchar(n): variable length
- date
- time
- timestamp: date + time
- year/month interval
- day/time interval

4.3.3 Constraints

- not NULL
- primary key
- unique
- foriegn key
- column or tuple check

Example 4.6. add a start date that must come before hire date

```
alter table Employee
add column StartDate date
add constraint hire_before_start
   check (HireDate <= StartDate);
```

4.3.4 Triggers

trigger: procedure execute by the db in response to table change

- event
- condition
- action

```
create trigger log_addr
after update of addr, phone on person
referencing OLD as o NEW as n
for each row
mode DB2SQL
when (o.status = 'VIP' or n.status = 'VIP)
   insert into VIPaddrhist(pid, oldaddr, oldphone,
      newaddr, newphone, user, modtime)
   values (o.pid, o.addr, o.phone,
      n.addr, n.phone, user, current timestamp)
```

5 Views

5.1 Definition

View: a relation whose instance is determined by other relations

- **Virtual**: views not stored, used only for querying
- **Materialized**: query for view is executed and view is stored

```
create [materialized] view <name>
   as query
```

Example 5.1. Manufacturing projects view

```
create view ManufacturingProjects as
   ( select projno, projname, firstname, lastname
     from project, employee
     where respemp = empno and deptno = 'D21' )
```

5.2 Updating

Changes to a view schema propogate back to instances of relations in conceptual schema, so to avoid ambiguity a view is updateable if:

- the query references exactly one table
- the query only outputs simple attributes
- there is **no** grouping/aggregation/distinct

- there are no nested queries
- there are no set operations

Materialized views also have to be update with periodically to account for base table changes

6 Application Development

6.1 Embedded SQL

6.1.1 Static Embedded SQL

Embed SQL into C with `EXEC SQL` and suffixing with `;`, using host variables to send and recieve values from DB

Example 6.1. Host variables in C

```
EXEC SQL BEGIN DECLARE SECTION;
char deptno[4];
char deptname[30];
char mgrno[7];
char admrdept[4];
char location[17];
EXEC SQL END DECLARE SECTION;

/ * program assigns values to variables * /

EXEC SQL INSERT INTO
   Department(deptno,deptname,mgrno,admrdept,location)
VALUES
   (:deptno,:deptname,:mgrno,:admrdept,:location);
```

indicator variables are flags used to handle host variables that might recieve `NULL`

Example 6.2. Indicator variables

```
int PrintEmployeePhone( char employeenum[] ) {
   EXEC SQL BEGIN DECLARE SECTION;
      char empno[7];
      char phonenum[5];
      short int phoneind;
   EXEC SQL END DECLARE SECTION;
      strcpy(empno,employeenum);
   EXEC SQL
      SELECT phoneno INTO :phonenum :phoneind
      FROM employee WHERE empno = :empno;
   if( SQLCODE < 0) { return( -1 ); } / * error * /
   else if(SQLCODE&=& 100){printf("no such employee\n");}
   else if (phoneind<0){printf("phone unknown\n");}
   else { printf("%s\n",phonenum); }
   return( 0 );
}]
```