CS 348: Intro to Database Management

Michael Noukhovitch

Winter 2015, University of Waterloo

Notes written from *'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);
```