Outline

- Why study and use database systems?
- Central concepts of database systems.
- Basic introduction to SQL.

Applications of Database Technology

Original

- inventory control
- payroll
- electronic funds transfer
- reservations systems

More recent

- computer aided design (CAD)
- computer aided software engineering (CASE)
- software development environments (SDE/SEE)
- telecommunications systems (AIN)
- the web!

Circumstances in Common

- Data is formatted.
- Data is important.
  - → Need to remember data reliably.
  - → Need to manipulate data easily.
- There are large amounts of data.
  - → Need to use mass store.
- Many users require simultaneous access to data.
  - → Need concurrency control.
- A large number of applications access the data.
  - → Need security.
  - → Need **data independence.**

Database Management

Basic idea

- Remove details related to data storage and access from application programs.
- Concentrate those functions in single subsystem: the **Database Management System** (DBMS).
- Have all applications access data through the DBMS.

Advantages

- Uncontrolled redundancy can be reduced.
- Less risk of inconsistency.
- Data integrity can be maintained.
- Access restrictions can be applied.
- Conflicting requirements can be balanced.

But most importantly

- a higher degree of **data independence** can be achieved.

Program Data Independence

Objective: to isolate application programs as much as possible from changes to data and to descriptions of data.

Two kinds of data independence

1. **physical data independence**
   (application programs immune to changes in storage structures)
2. **logical data independence**
   (application programs immune to changes in data descriptions)

Examples of changes in storage structures

- data encoding
- record structure
- file structure

Data dependence is expensive because changes in the way data is stored or described requires changes in application programs.

Brief History of Data Management

First generation (50's and 60's): files on tape

- batch processing
- sequential files on tape
- input on punched cards

Second generation (60's): files on disk

- disks enabled random access files
- new access methods (ISAM, hash files) were developed
- mostly batch with some interactive processing
- independent application systems with separate files
- growing application base

Brief History of Data Management (cont'd)

As the application base grows, we end up with

→ many shared files

→ a multitude of file structures

→ a need to exchange data between applications

This causes a variety of problems

- (redundancy) multiple copies
- (inconsistency) independent updates
- (inaccuracy) concurrent update mishandled
- (incompatibility) multiple formats, constraints
- (insecurity) proliferation
- (inaudability) poor chain of responsibility
- (inflexibility) changes difficult to apply

Brief History of Data Management (cont'd)

Third generation (mid 60's and 70's): early database systems

- beginning to separate between:
  logical view and physical implementation
- network model and hierarchical model introduced
- first batch oriented; on-line support added later
- transaction management added
  (concurrency control, recovery)
- access control facilities provided

Fourth generation (80's): relational technology

- simple, solid conceptual model
- strict separation of: logical view and physical implementation
- powerful, set-oriented query languages (SQL)
- distributed databases emerging

Brief History of Data Management (cont'd)

Fifth generation (90's): post-relational systems

- added functionality, more complex data (temporal, spatial)
- serving a broader class of applications
- object-oriented systems
- logic-based deductive systems
- active databases
- multidatabase systems

The Three-Schema Architecture

A **schema** (also called scheme) is a description of the data contents, structure, and possibly other aspects of the database.

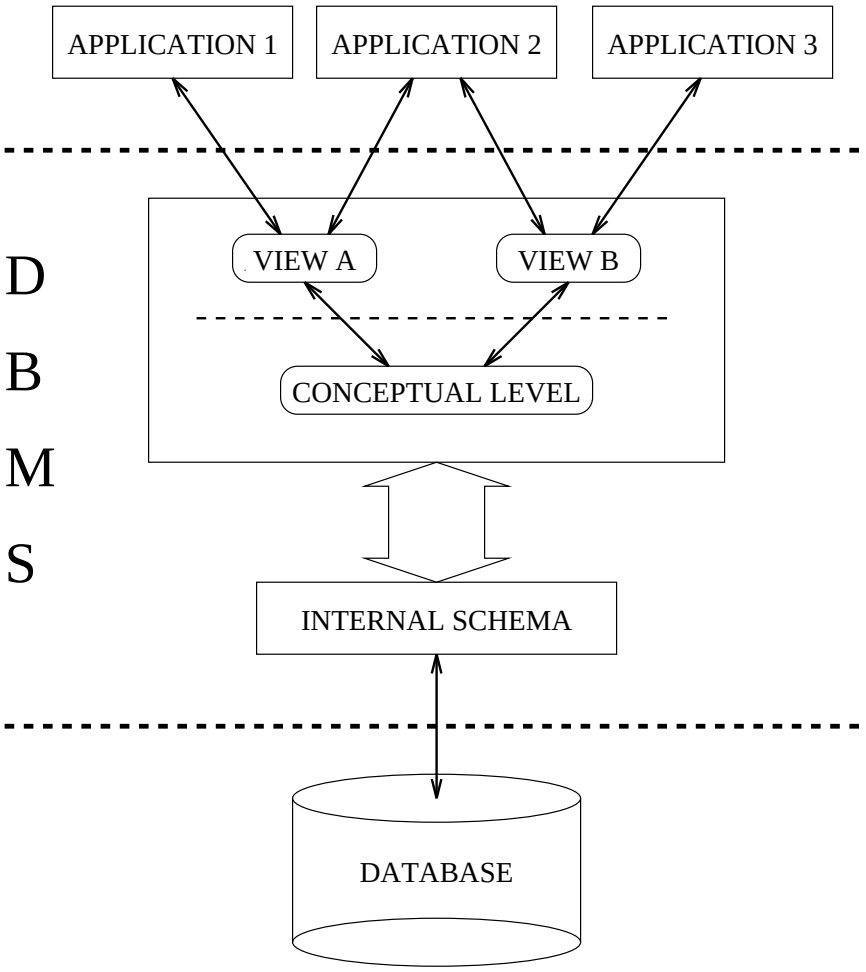
1. External schema (view): describes data as seen by an application program or by an end user.
2. Conceptual schema: describes the base logical structure of all data
3. Internal schema: describes how the database is physically encoded, including selection of files, indexes, etc.

Separation of external schema from conceptual schema enables logical data independence.

Separation of conceptual schema from internal schema enables physical data independence.

A database schema (or intention) *is different from* a database instance (or extension).

The Three-Schema Architecture (cont'd)



Interfacing to the DBMS

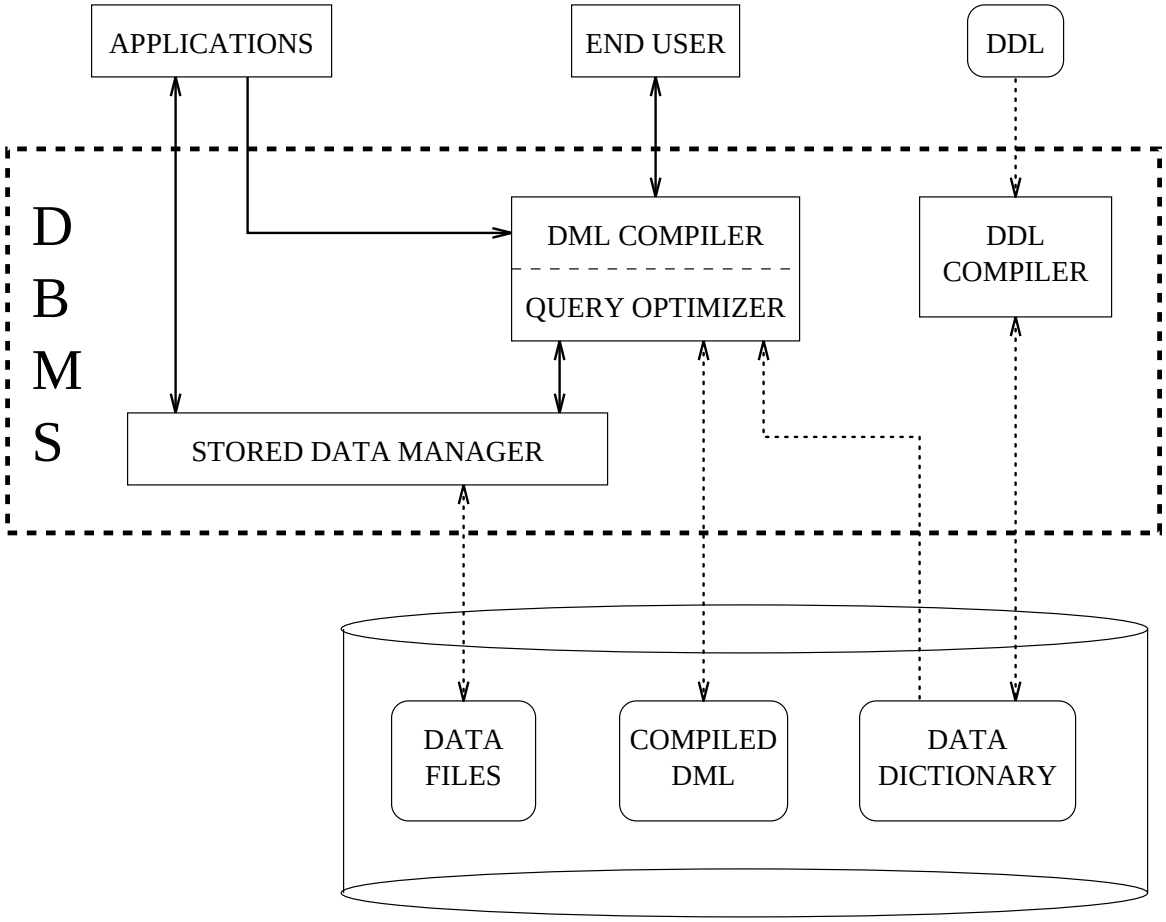
Data Definition Language (DDL)

- for specifying schemas
- may have different DDLs for (1) external schemas, (2) conceptual schemas, and (3) internal schemas
- information is stored in the **data dictionary**

Data Manipulation Language (DML)

- for specifying data queries and updates
- two general ways of querying and updating a database
  1. through "stand alone" DML facilities
  2. from within application programs
- two kinds of DMLs
  1. **navigational** (one record at a time)
  2. **non-navigational**

Components of a DBMS



Requirements for a DBMS

1. Provide data definition facilities
    - define a data definition language (DDL)
    - provide a user-accessible catalog (data dictionary) (database should be self-describing)
2. Provide facilities for storing, retrieving and updating data
    - define a data manipulation language (DML)
3. Support multiple views of data (user views)
    - end user or application should see only the data needed, and in form required

Requirements for a DBMS (cont'd)

4. Provide facilities for specifying integrity constraints (integrity constraint ⇔ update validation checks)
   - primary key constraints (identity integrity)
   - foreign key constraints (referential integrity)
   - more general constraints
5. Provide facilities for controlling access to data
   - prevent unauthorized access and update
6. Allow simultaneous access and update by multiple users
   - provide a concurrency control mechanism

Requirements for a DBMS (cont'd)

7. Support (logical) transactions
   - a sequence of operations to be performed as an atomic action
   - all operations are performed or none
   - equivalent to performing the operations instantaneously
8. Provide facilities for database recovery
   - can never lose the database, for whatever reason
   - bring the database back to a consistent state after a failure (disk failure, faulty program, earth quake, ...)
9. Provide facilities for database maintenance (utilities)
   - maintenance operations: redefine, unload, reload, mass insertion and deletion, validation, reorganization, ... (preferably without needing to shut down system)

Types of Users

End user

- (naive user) Accesses DBMS through menus.
- (sophisticated user) Writes ad-hoc queries using DML.

Application developer

- (programmer) Implements applications to access the database
  - using 3GL and embedded DML
  - using 4GL
- (analyst) Develops application specifications
  - using DDL to defined application views
  - using CASE tool

Database administrator (DBA)

Database system implementor

Role of a Database Administrator

- Manages conceptual schema.
- Assists with application view integration.
- Monitors overall performance of DBMS.
- Defines internal schema.
- Loads and reformats database.
- Is responsible for security and reliability.

Overview of SQL

Structured Query Language

Based on ISO 9075, an international standard for relational database systems.

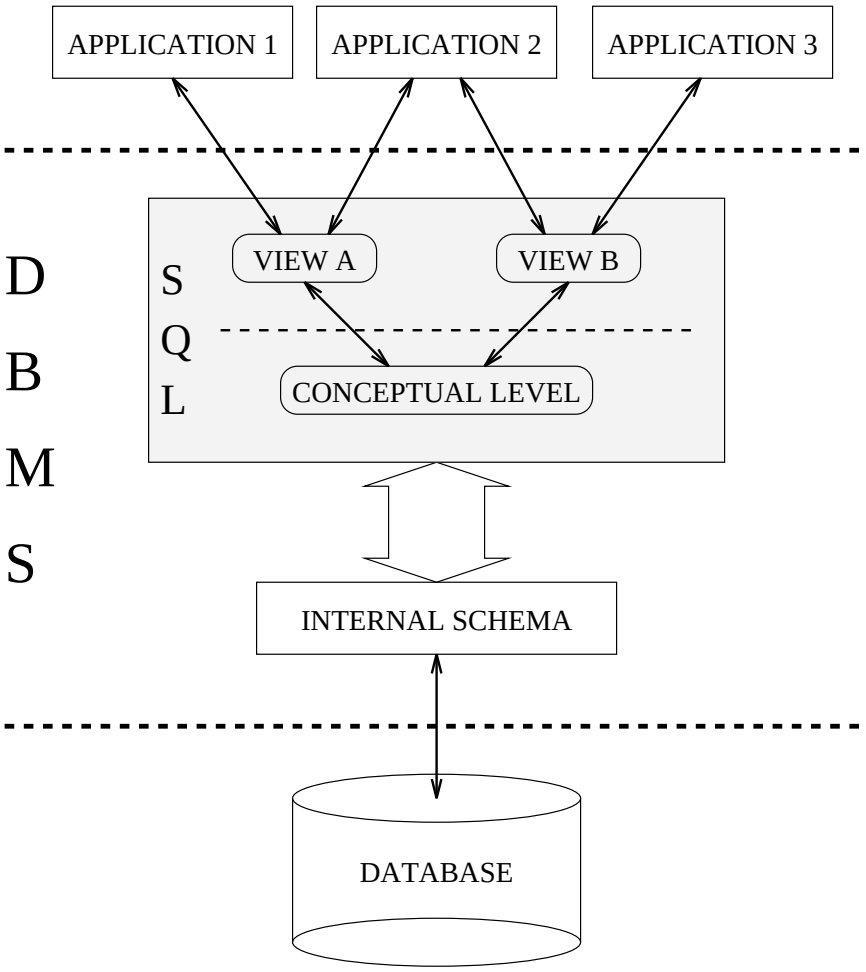
The standard is evolving

- (1986) Initial version.
- (1989) Most commercial products conform to this version.
- (1992) SQL2 (three levels of conformity $\approx$ 600 pages).
- (1999) SQL:1999 ($\approx$ 1200 pages; layered standard).

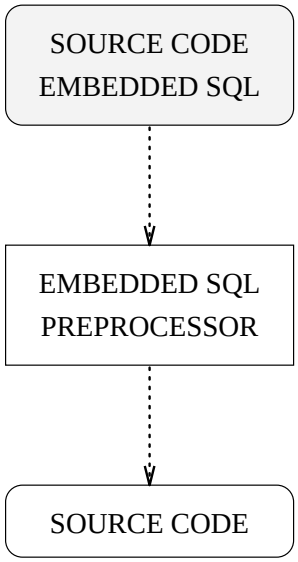
Main Features of the SQL Standard

- Powerful view definition language.
- Integrity constraints in conceptual schema.
- DML can be embedded in various programming languages.
- Authorization sublanguage/model.
- Transaction control.

Overview of SQL (cont'd)



Overview of SQL (cont'd)



Current languages supported: Ada, C, COBOL, Fortran, MUMPS, Pascal, PL/1.

Underlying Relational Model

Example relational database for a credit card company.

Vendor

Vno	Vname	City	Vbal
1	Sears	Toronto	200.00
2	Kmart	Ottawa	671.05
3	Esso	Montreal	0.00
4	Esso	Waterloo	2.25

Customer

AccNum	Cname	Prov	Cbal	Climit
101	Smith	Ont	25.15	2000
102	Jones	BC	2014.00	2500
103	Martin	Que	150.00	1000

Transaction

Tno	Vno	Acc#	Tdate	Amount
1001	2	101	940115	13.25
1002	2	103	940116	19.00
1003	3	101	940115	25.00
1003	4	102	940120	16.13
1004	4	103	940125	33.12

Structure of a Relational Database

Database: collection of uniquely named **tables** (**relations**).

Relation: set of **rows** (**tuples**).

Column = attribute.

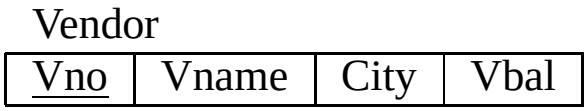
Domain: set of allowed values for an attribute.

Attribute values must be **atomic**: no tuples or sets or . . ..

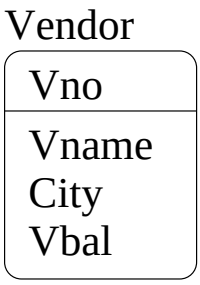
row $\sim$ distinguishable thing, or a relationship between a collection of things.

table $\sim$ set of related things or relationships.

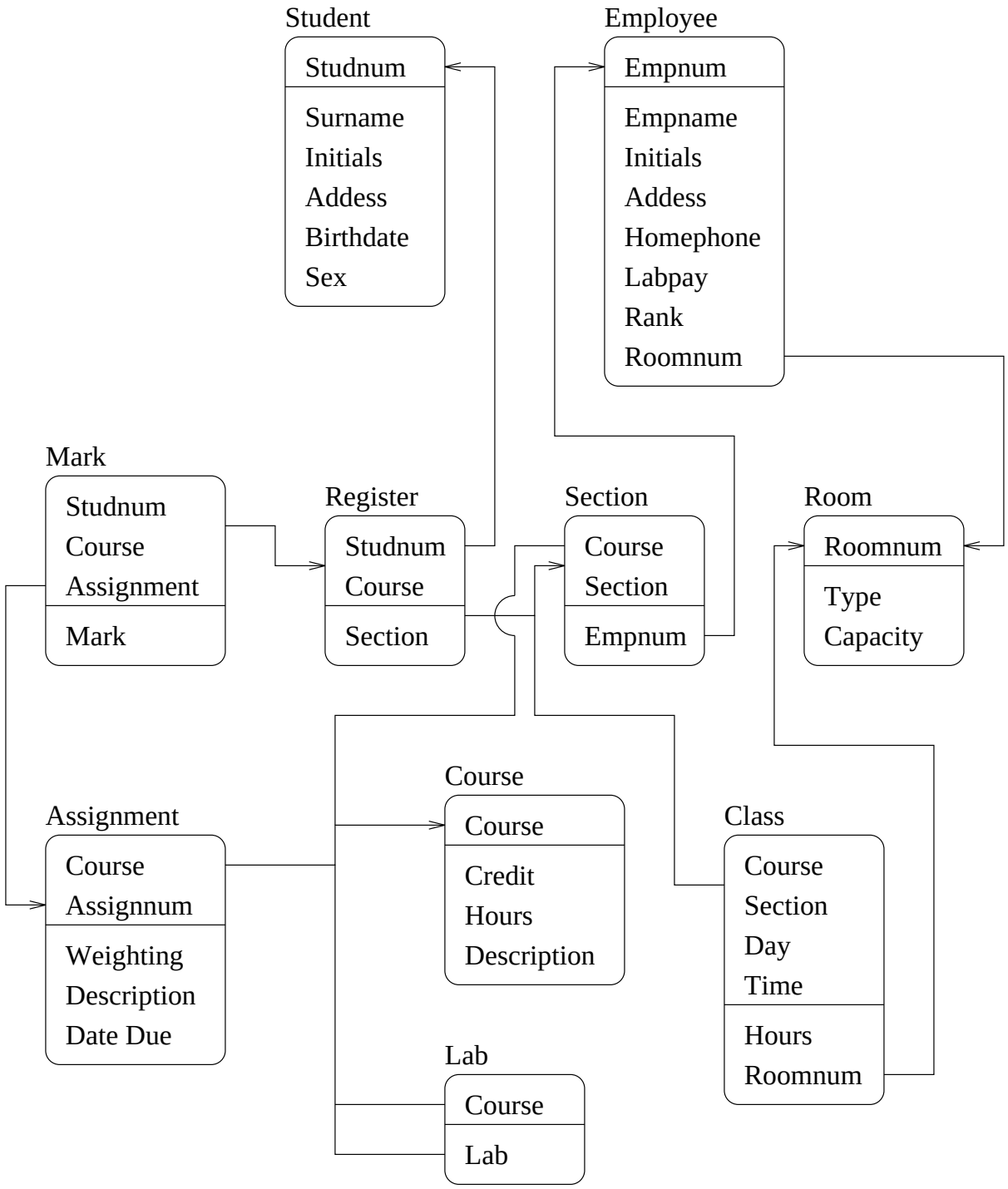
Diagrammatic Conventions



or



Managing Student Enrollment



The SQL DDL

Used for defining

- tables
- views

Tables (conceptual schema)

```
create table Vendor
  ( Vno          INTEGER not null ,
    Vname        VARCHAR(20),
    City         VARCHAR(10),
    Vbal         DECIMAL(10,2),
  primary key (Vno)   );
```

The SQL DDL (cont'd)

```
create table Customer
  ( AccNum     INTEGER not null ,
    Cname      VARCHAR(20) not null ,
    Prov       VARCHAR(20),
    Cbal       DECIMAL(6,2) not null ,
    Climit     DECIMAL(4,0) not null ,
  primary key (AccNum)   );

create table Transaction
  ( Tno        INTEGER not null ,
    Vno        INTEGER not null ,
    AccNum     INTEGER not null ,
    Day        DATE,
    Amount     DECIMAL(6,2) not null ,
  primary key (Tno),
  foreign key (Vno) references Vendor(Vno),
  foreign key (AccNum)
  references Customer(AccNum)   );
```

Attribute domains in SQL

INTEGER (integers representable with 32 bits)

SMALLINT (integers representable with 16 bits)

DECIMAL(m**,**n**)** (fixed point numbers)

FLOAT (32 bit floating point numbers)

CHAR(n**)** (fixed length strings)

VARCHAR(n**)** (variable length strings)

BIT(n**)** (n bits)

BIT VARYING(n**)** (variable number of bits)

Attribute domains in SQL (cont'd)

DATE (year, month, day)

TIME (hour, minute, second)

TIME(*i*) (hour, minute, second, second fraction)

TIMESTAMP (date, time, second fraction)

INTERVAL YEAR/MONTH (year month interval)

INTERVAL DAY/TIME (day time interval)

Can also declare a new domain.

create domain NAME **as** VARCHAR(20);

The SQL DDL (cont'd)

Views (external schema)

A view is a named query
(result is computed when the view is used)

```
create view WatVendors as
        select VNo, VName, VBal
        from Vendor
        where City = 'Waterloo';
```

Views can be used in retrieval exactly like tables (but updates of views are restricted).

Advantages of Views

- Logical data independence.
- Simplified perception of the database.
- Different views for different users.
- Restricting data access.

SQL has a Non-Navigational DML

Ex. "Find names and provinces of customers who owe more than $1000 to the company."

select Cname, Prov
from Customer
where Cbal $>$ 1000

The SQL DML

Basic querying

select *columns*
from $R_1, \ldots, R_k$
[**where** *filter*]

Result is a relation over *columns*
(*columns* = * means all attributes in $R_1, \ldots, R_k$)

$R_1, \ldots, R_k$: tables from which the data is retrieved

filter: conditions on tuples used to form the result

The SQL DML (cont'd)

Conditions may include

- arithmetic operators +, -, *, /
- comparisons $=$, $<>$, $<$, $<=$, $>$, $>=$
- logical connectives **and** , **or** and **not**

Ex. "List the names of the customers who live in Ontario and whose balance is over 80% of their balance limit."

select Cname
from Customer
where Prov = 'Ont' **and** Cbal $>$ 0.8 $*$ Climit

The SQL DML (cont'd)

Basic update

Insertion

insert into Customer
values (104, 'Grant', 'BC', 0, 4000)

Deletion

Delete Customer rows for customers named Smith:

delete from Customer
where Cname = 'Smith'

The SQL DML (cont'd)

Delete all transactions:

delete Transaction

Modification

Set balance of account 102 to zero:

update Customer **set** Cbal = 0
where AccNum = 102

Add $100 to each customer's monthly limit:

update Customer **set** Climit = Climit + 100