CSCC43H: Introduction to Databases

Lecture 4

Wael Aboulsaadat

Acknowledgment: these slides are partially based on Prof. Garcia-Molina & Prof. Ullman slides accompanying the course's textbook.

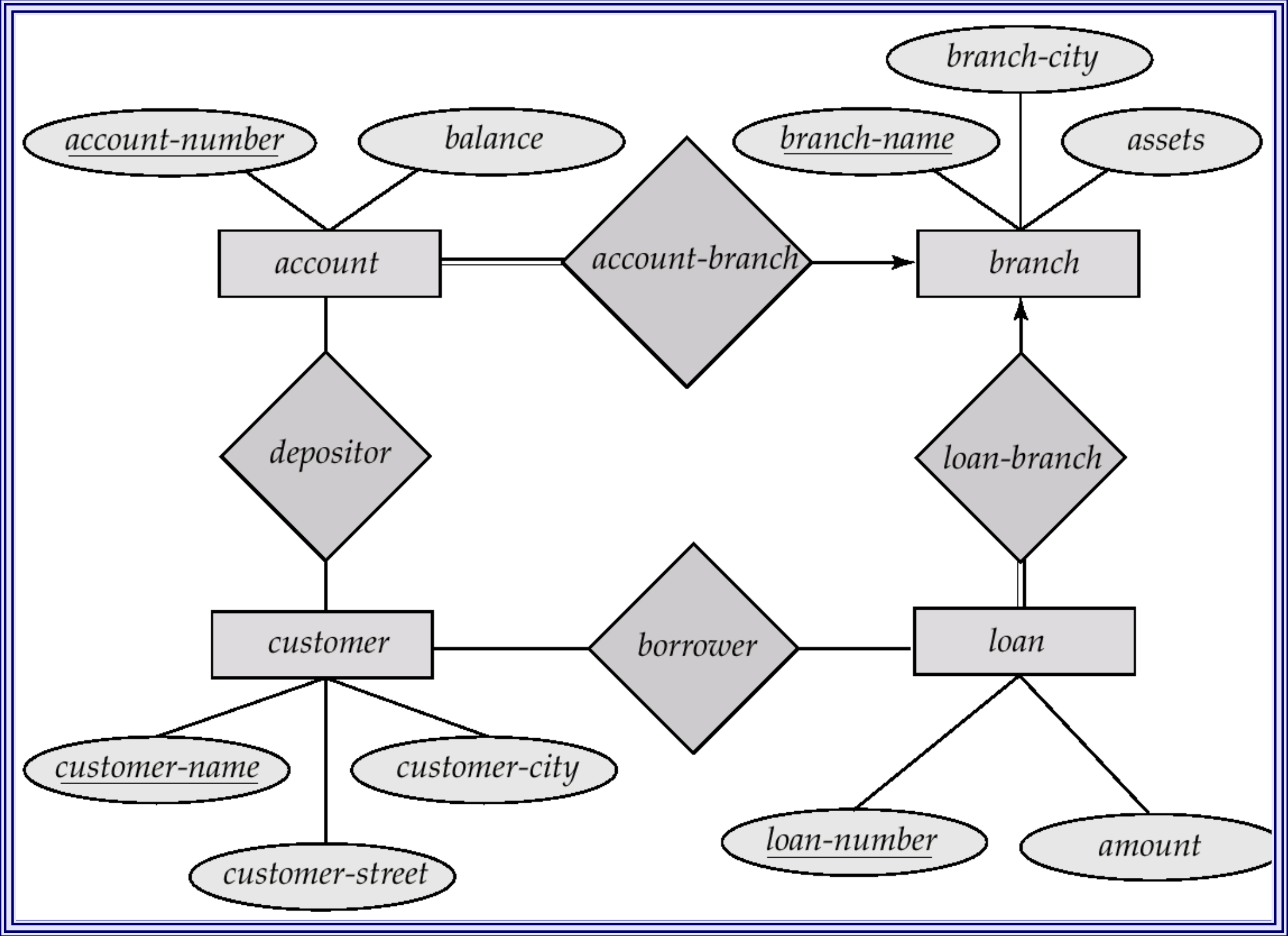
Database Management System (DBMS)

- A collection of programs that enable:
  - ➔Defining (describing the structure),
  - – Populating by data (Constructing),
  - – Manipulating (querying, updating),
  - – Preserving consistency,
  - – Protecting from misuse,
  - – Recovering from failure, and
  - – Concurrent using

  of a database.

Banking Example

E-R Diagram for the Banking Enterprise



Banking Example

branch (branch_name, branch_city, assets)

customer (customer_name, customer_street, customer_city)

account (account_number, branch_name, balance)

loan (loan_number, branch_name, amount)

depositor (customer_name, account_number)

borrower (customer_name, loan_number)

Example Queries

- Find all loans of over $1200

$$\sigma_{amount > 1200}\ (loan)$$

- Find the loan number for each loan of an amount greater than $1200

$$\Pi_{loan\_number}\ (\sigma_{amount > 1200}\ (loan))$$

- Find the names of all customers who have a loan, an account, or both, from the bank

$$\Pi_{customer\_name}\ (borrower) \cup \Pi_{customer\_name}\ (depositor)$$

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.

$$\Pi_{customer\_name} (\sigma_{branch\_name=\text{"Perryridge"}} (\sigma_{borrower.loan\_number = loan.loan\_number}(borrower \times loan)))$$

- Find the names of all customers who have a loan at the Perryridge branch but did not deposit at any branch of the bank.

$$\Pi_{customer\_name} (\sigma_{branch\_name = \text{"Perryridge"}} (\sigma_{borrower.loan\_number = loan.loan\_number}(\text{borrower x loan})))\ - \Pi_{customer\_name}(\text{depositor})$$

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.
  - Query 1

$$\Pi_{customer\_name} (\sigma_{branch\_name = \text{“Perryridge”}} (\sigma_{borrower.loan\_number = loan.loan\_number} (borrower \times loan)))$$

  - Query 2

$$\Pi_{customer\_name}(\sigma_{loan.loan\_number = borrower.loan\_number} ((\sigma_{branch\_name = \text{“Perryridge”}} (loan)) \times borrower))$$

Example Queries

- Find the largest account balance
  - Strategy:
    - Find those balances that are *not* the largest
      - Rename *account* relation as *d* so that we can compare each account balance with all others
    - Use set difference to find those account balances that were *not* found in the earlier step.
  - The query is:

$$\Pi_{balance}(account) - \Pi_{account.balance}$$

$$(\sigma_{account.balance < d.balance}(account \times \rho_d(account)))$$

Bank Example Queries

- **Find the names of all customers who have a loan and an account at bank.**

$$\Pi_{customer\_name}(borrower) \cap \Pi_{customer\_name}(depositor)$$

- Find the name of all customers who have a loan at the bank and the loan amount

$$\Pi_{customer\_name,\ loan\_number,\ amount}(borrower \bowtie loan)$$

Bank Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.

Bank Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.

$$\Pi_{customer\_name,\ branch\_name}(depositor \bowtie account) \div \Pi_{branch\_name}(\sigma_{branch\_city = \text{“Brooklyn”}}(branch))$$

Example 2

Given relational schema:

Sailors (sid, sname, rating, age)

Reservation (sid, bid, date)

Boats (bid, bname, color)

1) Find names of sailors who've reserved boat #103
2) Find names of sailors who've reserved a red boat
3) Find sailors who've reserved a red or a green boat
4) Find sailors who've reserved a red <u>and</u> a green boat
5) Find the names of sailors who've reserved all boats

Structured Query Language (SQL)

Structure Query Language

Data Definition Language (DDL)	Data Manipulation Language (DML)

Data Definition Language (DDL)

Allows the specification of not only a set of relations but also information about each relation, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints (what’s valid….)
- The set of indices (keys..) to be maintained for each relations.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Domains

- Domains specify allowable values for attributes.
- Two categories:
  - Elementary (predefined by the standard);
  - User-defined.

Elementary Domains — Character

- Character
  - Single characters or strings;
  - Strings may be of variable length;
  - A Character set different from the default one can be used (e.g., Latin, Greek, Cyrillic, etc.)
  - Syntax:

    character [varying] [(*Length*)] [character set *CharSetName*]
  - It is possible to use **char** and **varchar**, for **character** and **character varying** respectively

More Elementary Domains

- Bit
  - Single Boolean values or strings of Boolean values (may be variable in length);
  - Syntax:

    bit [**varying**] [(*Length*)]
- Exact numeric domains
  - Exact values, integer or with a fractional part
  - Four alternatives: numeric(6,3)

    numeric [(*Precision* [, *Scale*])]
    decimal [(*Precision* [, *Scale*])]
    integer
    smallint

# of significant digits *decimal digits*

Approximate Numeric Domains

- Approximate numeric domains
  - Approximate real values
  - Based on a floating point representation

```
float [ ( Precision ) ]
double precision
```

Temporal Instant Domains

- Temporal instants

  date has fields **year,month,day**

  time [(*Precision*)] [**with time zone**]

  has fields **hour,minute,second**

  timestamp [(*Precision*)] [**with time zone**]

- Temporal intervals

  interval *FirstUnitOfTime* [**to** *LastUnitOfTime*]

  – Units of time are divided into two groups:
    - (i) year, month,
    - (ii) day, hour, minute, second
  – For example, **year(5) to month** allows intervals up to **99999yrs + 11mo**

User-Defined Domains

- Comparable to definitions of variable types in programming languages.
- A domain is characterized by name, elementary domain, default value, set of constraints
- Syntax:

create domain *DomainName*

as *ElementaryDomain* [*DefaultValue*] [*Constraints*]

- Example:

```
create domain Mark as smallint default null
```

Default Domain Values

- Define the value that the attribute must assume when a value is not specified during row insertion.
- Syntax:

  `default` < ***GenericValue*** | **`user`** | **`null`** >

- ***GenericValue*** represents a value compatible with the domain, in the form of a constant or an expression.
- **`user`** is the login name of the user who assigns a value to this attribute.

Summary: domain types in SQL

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with *n* digits to the right of decimal point.
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.
- Null values are allowed in all the domain types. Declaring an attribute to be **not null** prohibits null values for that attribute.
- **create domain** construct in SQL-92 creates user-defined domain types **create domain** person-name **char**(20) **not null**

Summary: domain types in SQL (cont.)

- **date.** Dates, containing a (4 digit) year, month and date
  - E.g. **date** '2001-7-27'
- **time.** Time of day, in hours, minutes and seconds.
  - E.g. **time** '09:00:30' **time** '09:00:30.75'
- **timestamp**: date plus time of day
  - E.g. **timestamp** '2001-7-27 09:00:30.75'
- **Interval**: period of time
  - E.g. Interval '1' day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values
- Can extract values of individual fields from date/time/timestamp
  - E.g. **extract** (**year** **from** r.starttime)
- Can cast string types to date/time/timestamp
  - E.g. **cast** <string-valued-expression> **as** **date**

Schema Definition

- A schema is a collection of objects: domains, tables, indexes, assertions, views, privileges
- A schema has a name and an owner (who determines authorization privileges)
- Syntax:

  `create schema` [*SchemaName*]

  [[**`authorization`**] *Authorization*] { *SchemaElementDefinition* }

Table Definition

- An SQL table consists of an ordered set of attributes, and a (possibly empty) set of constraints
- Statement **`create table`** defines a relation schema, creating an empty instance.
- Syntax:

  `create table` *TableName*

  (*AttributeName Domain* [*DefaultValue*] [*Constraints*]
  {, *AttributeName Domain* [*DefaultValue*] [*Constraints*] }
  [*OtherConstraints*])

Create Table Construct

- An SQL relation is defined using the **create table** command:

  create table *r* (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - r is the name of the relation
  - each A_i is an attribute name in the schema of relation r
  - D_i is the data type of values in the domain of attribute A_i

Example of `create table`

```
Employee:
     RegNo     is 6 characters
     FirstName is 20 characters
     Surname   is 20 characters
     Salary    is 9 numeric
     City      is 15 characters
```

Example of `create table`

```
create table Employee
  (
     RegNo     character(6),
     FirstName character(20),
     Surname   character(20),
     Salary    numeric(9),
     City      character(15)
  )
```

Intra-Relational Constraints

- Constraints are conditions that must be verified by every database instance
- Intra-relational constraints involve a single relation
  - **`not null`** (on single attributes)
  - **`unique`**: permits the definition of keys; syntax:
    - for single attributes: **`unique`**, after the domain
    - for multiple: **`unique`** **(*Attribute* {, *Attribute* })**
  - **`primary key`**: defines the primary key (once for each table; implies **not null**); syntax like **unique**
  - `check`: described later

Example of Intra-Relational Constraints

- Each pair of **`FirstName`** and **`Surname`** uniquely identifies each element

```
FirstName char(20) not null,
Surname char(20) not null,
unique(FirstName,Surname)
```

Inter-Relational Constraints

Constraints may involve several relations:

- **`check`**: checks whether an assertion is true;
- **`references`** and **`foreign key`** permit the definition of referential integrity constraints;
  - Syntax for single attributes
    `references` after the domain
  - Syntax for multiple attributes
    `foreign key` **(*Attribute* {, *Attribute* })**
    `references` …
- It is possible to associate reaction policies to violations of referential integrity constraints.

Example

```
create table Employee
(
    RegNo char(6),
    FirstName char(20) not null,
    Surname char(20) not null,
    Dept char(15),
    Salary numeric(9) default 0,
    City char(15),
    primary key(RegNo),
    foreign key(Dept) references Department(DeptName),
    unique(FirstName,Surname)
)
```