Schema Design and SQL Join
Database – web application perspective

- Database
  - data model
  - transactional storage (ACID)
    - application requirements, e.g. financial charge
  - computation

- Most web applications mainly take the DB as the transactional storage
  - many emerging systems, e.g. distributed systems/data center
  - different applications may need different level of consistency
    - ACID is a very strong requirement
  - performance & cost tradeoff
Database Design

- Web applications may require complex data model
- Schema design
  - or say: what kind of tables I need?
  - also expected in the first meeting (but not required)
- General steps
  - First: E/R model
  - Second: further to relational tables
Problem description

- The bank has multiple branches
  - Branch info: name, address, asset
- The bank has many customers
  - Customer: name, address
- Each customer can have arbitrary accounts
  - Account: account number
- Two types of accounts:
  - Checking: overdraft amount
  - Saving: rate
- Each customer can apply the loan from ONE branch
  - Loan: amount
E/R model, step 1: entities & attributes

- The bank has multiple branches
  - Branch info: name, address, asset
- The bank has many customers
  - Customer: name, address
- Each customer can have arbitrary accounts
- Two types of accounts:
  - Checking: overdraft amount
  - Saving: rate
- Each customer can apply the loan from ONE branch
  - Loan: amount
step 2: relationships

- The bank has multiple branches
  - Branch info: name, address, asset
- The bank has many customers
  - Customer: name, address
- Each customer can have arbitrary accounts
- Two types of accounts:
  - Checking: overdraft amount
  - Saving: rate
- Each customer can apply the loan from ONE branch
  - Loan: amount
E/R model
Step 1: Entities

- Entities that can be uniquely identified by its own attribute(s), e.g.
  - \( \text{customer\_id} \) identifies customer
  - \( \text{account\_num} \) identifies account

- Four tables
  - \( \text{branch}=(\text{branch\_name}, \text{branch\_city}, \text{assets}) \)
  - \( \text{customer}=(\text{customer\_id}, \text{customer\_name}, \text{customer\_addr}) \)
  - \( \text{loan}=(\text{loan\_number}, \text{amount}) \)
  - \( \text{account}=(\text{account\_num}, \text{balance}) \)
Step 2: Many-to-many relationship

- Two tables: attributes are primary keys of participating entities
  - borrower = (customer_id, loan_number)
  - depositor = (customer_id, account_number)
Step 3: Many-to-one relationship

- Two options:
  - A new table for the relationship
  - Add extra attribute(s) to “many” side

- New table
  - \textit{loan\_branch}(loan\_number, \textit{branch\_name})

- Add an extra attribute to \textit{loan}
  - \textit{loan}=(loan\_number, amount, \textit{branch\_name})
Step 4: ISA relationship

- Two tables
  - $savings\_account=(account\_number, interest\_rate)$
  - $checking\_account=(account\_number, overdraft\_amount)$
Step 5: Identify primary key & foreign key
Join: connect records from different tables

- Equal-join
  - condition is “="
  - In many cases, they are primary key-foreign key relationships

- Non-equal join
  - condition is “<” “>” or “<>”
Equal-join (primary key-foreign key)

Find all the customers who have loan account in the branch “Gilman Dr”

- **SELECT** c.customer_name
- **FROM** customer c, borrow b, loan l
- **WHERE** c.customer_id = b.customer_id AND b.loan_number = l.loan_number AND l.branch_name = ‘Gilman Dr’
Equal Join (ordinary attributes)

- **Schema**
  - Candidate(*cid*, *name*, *expect_salary*, *city*)
  - Job(*jid*, *company_name*, *salary*, *city*)

- **Find candidate-job pairs such that they are in the same city and job’s salary is at most 60k**
  - SELECT *
  - FORM candidate c, job j
  - WHERE c.city = j.city AND j.salary <= 60
Non-equal Join

- **Schema**
  - Candidate(`cid`, `name`, `expect_salary`, `city`)
  - Job(`jid`, `company_name`, `salary`, `city`)

- Find candidate-job pairs such that candidate’s expectation salary is lower than job’s
  - SELECT *
  - FROM candidate c, job j
  - WHERE c.expect_salary < j.salary
backup
Primary key

- $\text{customer}=(\text{customer}_\text{id}, \text{customer}_\text{name}, \text{customer}_\text{addr})$
  - Two rows $t1, t2$ in the $\text{customer}$ table
  - If $t1.\text{customer}_\text{id} = t2.\text{customer}_\text{id}$.
    - then $t1.\text{customer}_\text{name} = t1.\text{customer}_\text{name}$ AND $t1.\text{customer}_\text{addr} = t2.\text{customer}_\text{addr}$
Foreign key

- Foreign key identifies a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table.
- The values in one row of the referencing columns must occur in a single row in the referenced table.
- Example
  - `customer=(customer_id, customer_name, customer_addr)`
  - `borrower=(customer_id, loan_number)`
  - `borrower.customer_id` must exist in `customer`