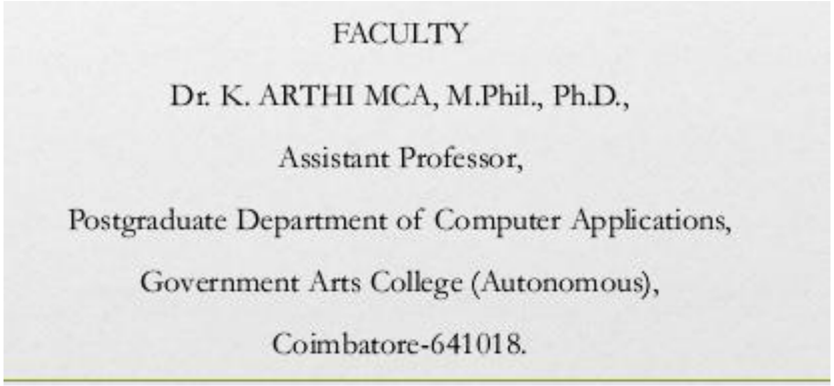
20MCA12C RELATIONAL DATABASE MANAGEMENT SYSTEM

UNIT IV: Relational Databases



FACULTY

Dr. K. ARTHI MCA, M.Phil., Ph.D.,

Assistant Professor,

Postgraduate Department of Computer Applications,

Government Arts College (Autonomous),

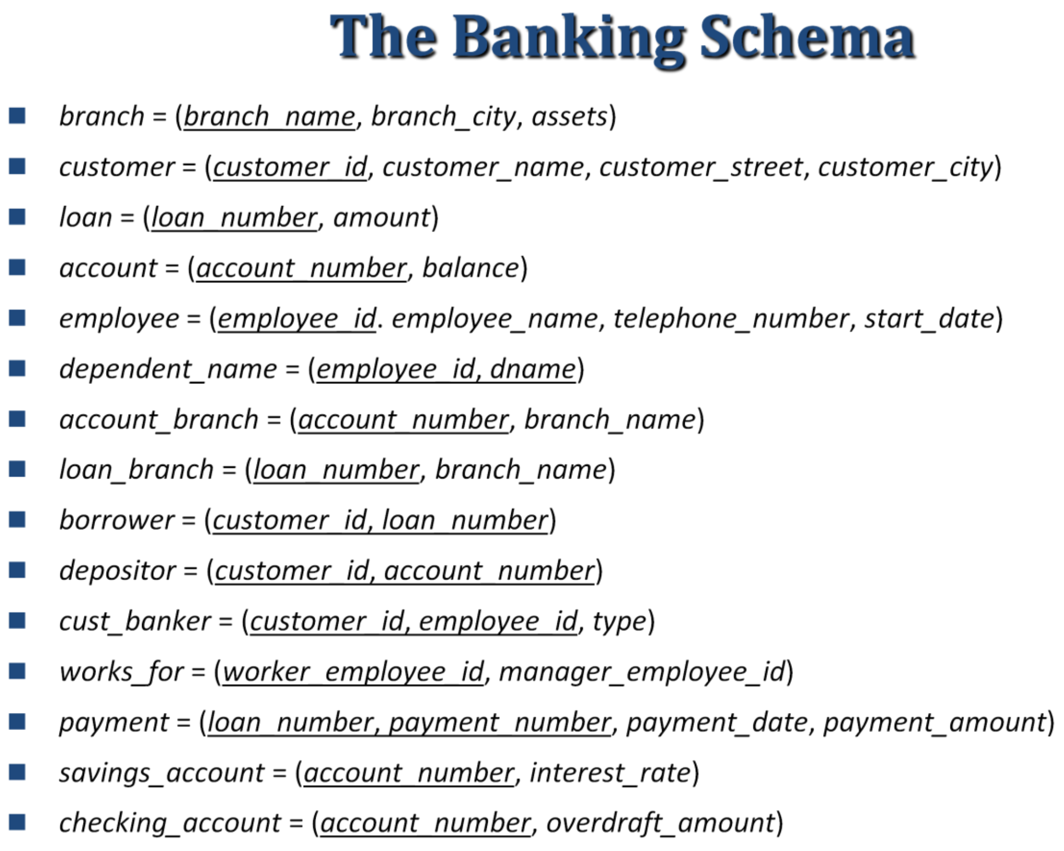
Coimbatore-641018.

Features of good relational designs

Design alternatives

1)larger schemas

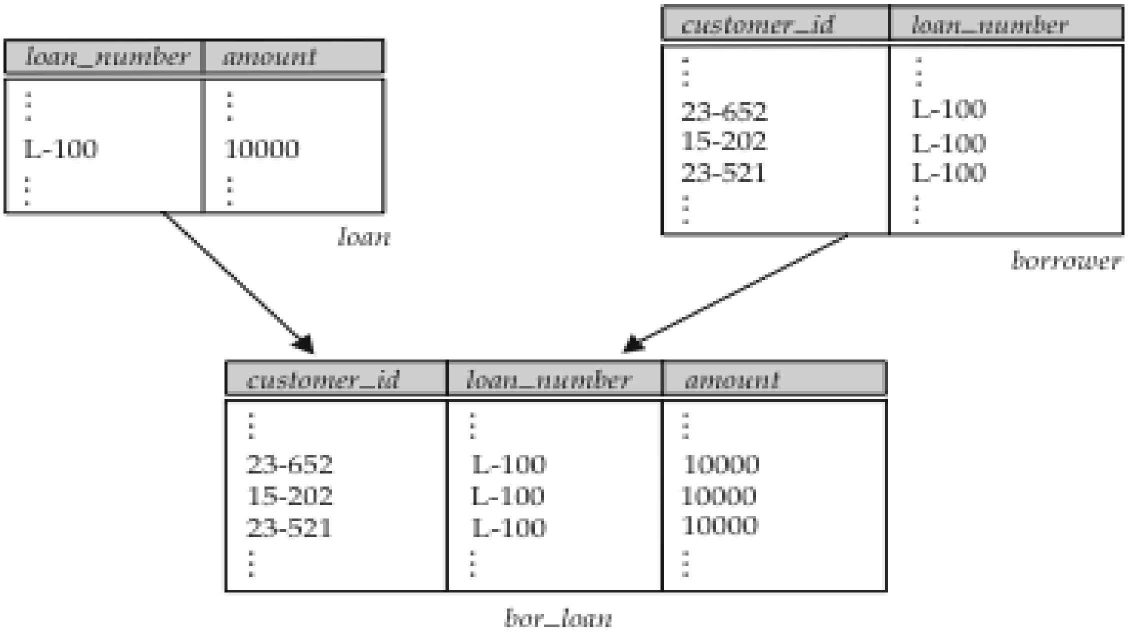
2)smaller schemas



The Banking Schema

- *branch* = (*branch_name*, *branch_city*, *assets*)
- *customer* = (*customer_id*, *customer_name*, *customer_street*, *customer_city*)
- *loan* = (*loan_number*, *amount*)
- *account* = (*account_number*, *balance*)
- *employee* = (*employee_id*. *employee_name*, *telephone_number*, *start_date*)
- *dependent_name* = (*employee_id, dname*)
- *account_branch* = (*account_number*, *branch_name*)
- *loan_branch* = (*loan_number*, *branch_name*)
- *borrower* = (*customer_id, loan_number*)
- *depositor* = (*customer_id, account_number*)
- *cust_banker* = (*customer_id, employee_id*, *type*)
- *works_for* = (*worker_employee_id*, *manager_employee_id*)
- *payment* = (*loan_number, payment_number*, *payment_date*, *payment_amount*)
- *savings_account* = (*account_number*, *interest_rate*)
- *checking_account* = (*account_number*, *overdraft_amount*)

combined schemas



loan_number	*amount*
⋮	⋮
L-100	10000
⋮	⋮

loan

customer_id	*loan_number*
⋮	⋮
23-652	L-100
15-202	L-100
23-521	L-100
⋮	⋮

borrower

customer_id	*loan_number*	*amount*
⋮	⋮	⋮
23-652	L-100	10000
15-202	L-100	10000
23-521	L-100	10000
⋮	⋮	⋮

bor_loan

Suppose we combine *borrower* and *loan* to get

***bor_loan* = (*customer_id*, *loan_number*, *amount*)**

Result is possible repetition of information

Smaller schemas

- Suppose we had started with *bor_loan.* How would we know to split up (**decompose**) it into *borrower* and *loan*?
- Write a rule "if there were a schema (*loan_number, amount*), then *loan_number* would be a candidate key"
- Denote as a **functional dependency**:

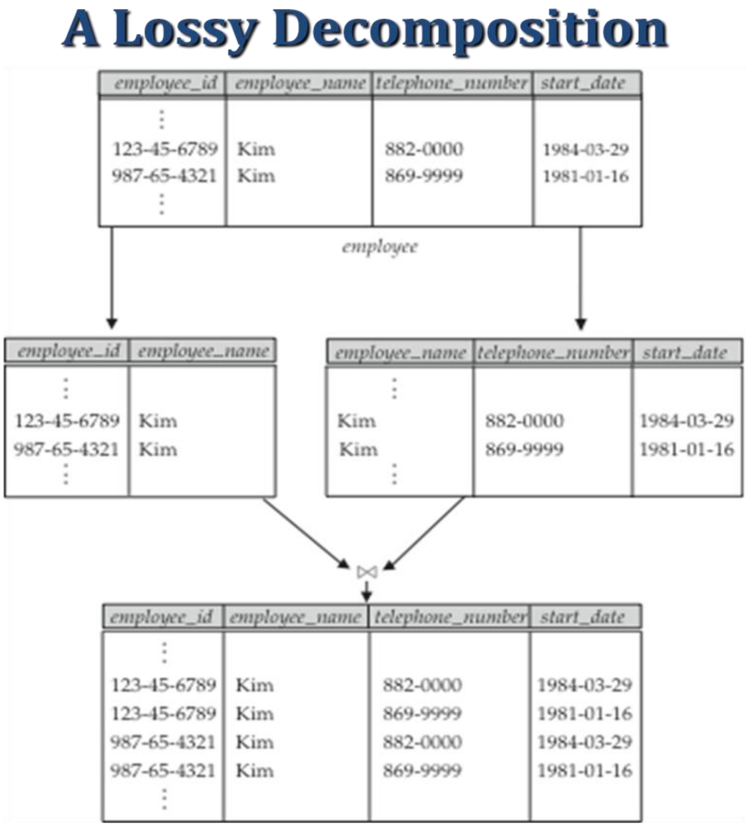
  $loan\_number \rightarrow amount$

- In *bor_loan*, because *loan_number* is not a candidate key, the amount of a loan may have to be repeated. This indicates the need to decompose *bor_loan*.
- Not all decompositions are good. Suppose we decompose *employee* into

  employee1 = (*employee_id*, *employee_name*)

  employee2 = (*employee_name*, *telephone_number*, *start_date*)

- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a lossy decomposition.



A Lossy Decomposition

employee_id	employee_name	telephone_number	start_date
⋮			
123-45-6789	Kim	882-0000	1984-03-29
987-65-4321	Kim	869-9999	1981-01-16
⋮			

employee

employee_id	employee_name
⋮	
123-45-6789	Kim
987-65-4321	Kim
⋮	

employee_name	telephone_number	start_date
⋮		
Kim	882-0000	1984-03-29
Kim	869-9999	1981-01-16
⋮		

employee_id	employee_name	telephone_number	start_date
⋮			
123-45-6789	Kim	882-0000	1984-03-29
123-45-6789	Kim	869-9999	1981-01-16
987-65-4321	Kim	882-0000	1984-03-29
987-65-4321	Kim	869-9999	1981-01-16
⋮			

First Normal Form

- Domain is atomic if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in first normal form if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account
  - We assume all relations are in first normal form

- Atomicity is actually a property of how the elements of the domain are used.
  - Example: Strings would normally be considered indivisible
  - Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
  - If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
  - Doing so is a bad idea: leads to encoding of information in application program rather than in the database.

FUNCTIONAL DEPENDENCY

A *functional dependency* (FD) is a relationship between two attributes, typically between the PK and other non-key attributes within a table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y. This relationship is indicated by the representation below :

X ———> Y

The left side of the above FD diagram is called the *determinant*, and the right side is the *dependent*.

- **Constraints on the set of legal relations.**
- **Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.**
- **A functional dependency is a generalization of the notion of a *key*.**

- Let *R* be a relation schema

$$\alpha \subseteq R \;\; and \;\; \beta \subseteq R$$

- The functional dependency

$$\alpha \rightarrow \beta$$

holds on *R* if and only if for any legal relations *r*(R), whenever any two tuples t_1 and t_2 of *r* agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \;\; \Rightarrow \;\; t_1[\beta] = t_2[\beta]$$

- Example: Consider *r*(A,*B*) with the following instance of *r.*

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  bor_loan = (*<u>customer_id, loan_number</u>, amount*).

  We expect this functional dependency to hold:

  $$loan\_number \rightarrow amount$$

  but would not expect the following to hold:

  $$amount \rightarrow customer\_name$$

- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
    - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* satisfies *F.*
  - specify constraints on the set of legal relations
    - We say that *F* holds on *R* if all legal relations on *R* satisfy the set of functional dependencies *F.*
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of *loan* may, by chance, satisfy *amount* → *customer_name.*

A **functional dependency is trivial if it is satisfied by all instances of a relation**

Example*:*

- ***customer_name, loan_number* → *customer_name***
- ***customer_name* → *customer_name***

In general, $\alpha \rightarrow \beta$ **is trivial if** $\beta \subseteq \alpha$

Boyce-Codd Normal Form

A relation schema *R* is in BCNF with respect to a set *F* of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for *R*

Example schema *not* in BCNF:

bor_loan = (*customer_id, loan_number, amount*)

because *loan_number* → *amount* holds on *bor_loan* but *loan_number* is not a superkey

Decomposing a Schema into BCNF

- Suppose we have a schema *R* and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

  We decompose *R* into:
  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$
- In our example,
  - α = *loan_number*
  - β = *amount*

  and *bor_loan* is replaced by
  - $(\alpha \cup \beta)$ = (*loan_number, amount*)
  - $(R - (\beta - \alpha))$ = (*customer_id, loan_number*)

BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving.*
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form.*

Third Normal Form

- A relation schema *R* is in third normal form (3NF) if for all:

  $$\alpha \rightarrow \beta \text{ in } F^+$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for *R*
  - Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R.*

  (**NOTE***:* each attribute may be in a different candidate key)
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).

Functional-Dependency Theory

- consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
- Then develop algorithms to generate lossless decompositions into BCNF and 3NF
- Then develop algorithms to test if a decomposition is dependency-preserving

Lossless-join Decomposition

- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless join if and only if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i.
  - A decomposition is dependency preserving, if $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$
  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.

Testing for Dependency Preservation

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into R_1, R_2, …, R_n we apply the following test (with attribute closure done with respect to F)

  -
    ```
    result = α
    while (changes to result) do
      for each R_i in the decomposition
            t = (result ∩ R_i)^+ ∩ R_i
            result = result ∪ t
    ```
  - If *result* contains all attributes in β, then the functional dependency $\alpha \rightarrow \beta$ is preserved.
- We apply the test on all dependencies in F to check if a decomposition is dependency preserving
- This procedure takes polynomial time, instead of the exponential time required to compute F^+ and $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

BCNF Decomposition Algorithm

```
result := {R };
done := false;
compute $F^+$;
while (not done) do
   if (there is a schema $R_i$ in result that is not in BCNF)
      then begin
               let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that holds on $R_i$
                        such that $\alpha \rightarrow R_i$ is not in $F^+$,
                        and $\alpha \cap \beta = \varnothing$;
                 result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
      end
      else done := true;
```

Note: each R_i is in BCNF, and decomposition is lossless-join.

Example of BCNF Decomposition

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF ($B \rightarrow C$ but B is not superkey)
- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A,B)$

Third Normal Form

- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems; we will see examples later)
  - But functional dependencies can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.

3NF Decomposition Algorithm

```
Let $F_c$ be a canonical cover for $F$;
$i$ := 0;
for each functional dependency $\alpha \rightarrow \beta$ in $F_c$ do
  if none of the schemas $R_j$, $1 \leq j \leq i$ contains $\alpha\beta$
        then begin
              $i := i + 1$;
              $R_i := \alpha\beta$
          end
if none of the schemas $R_j$, $1 \leq j \leq i$ contains a candidate key for $R$
  then begin
          $i := i + 1$;
          $R_i$ := any candidate key for $R$;
        end
return ($R_1$, $R_2$, ..., $R_i$)
```

Above algorithm ensures:

each relation schema R_i is in 3NF

decomposition is dependency preserving and lossless-join

Multivalued Dependencies (MVDs)

- Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The *multivalued dependency*

  $$\alpha \rightarrow\rightarrow \beta$$

  holds on R if in any legal relation $r(R)$, for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that:

  $$\begin{aligned} t_1[\alpha] = t_2[\alpha] &= t_3[\alpha] = t_4[\alpha] \\ t_3[\beta] &= t_1[\beta] \\ t_3[R - \beta] &= t_2[R - \beta] \\ t_4[\beta] &= t_2[\beta] \\ t_4[R - \beta] &= t_1[R - \beta] \end{aligned}$$

$\rightarrow\rightarrow$

Fourth Normal Form

- A relation schema R is in 4NF with respect to a set D of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
  - $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  - α is a superkey for schema R
- If a relation is in 4NF it is in BCNF

$\rightarrow\rightarrow$

4NF Decomposition Algorithm

```
result: = {$R$};
  done := false;
  compute $D^+$;
  Let $D_i$ denote the restriction of $D^+$ to $R_i$
while (not done)
    if (there is a schema $R_i$ in result that is not in 4NF) then
      begin
        let $\alpha \rightarrow\rightarrow \beta$ be a nontrivial multivalued dependency that holds
        on $R_i$ such that $\alpha \rightarrow R_i$ is not in $D_i$, and $\alpha \cap \beta = \phi$;
       result := (result - $R_i$) $\cup$ ($R_i$ - $\beta$) $\cup$ ($\alpha$, $\beta$);
      end
    else done:= true;
```

Note: each R_i is in 4NF, and decomposition is lossless-join

$\rightarrow\!\!\rightarrow$

THANK YOU

This content is taken from the text books and reference books prescribed in the syllabus.