Data Base Lab-2

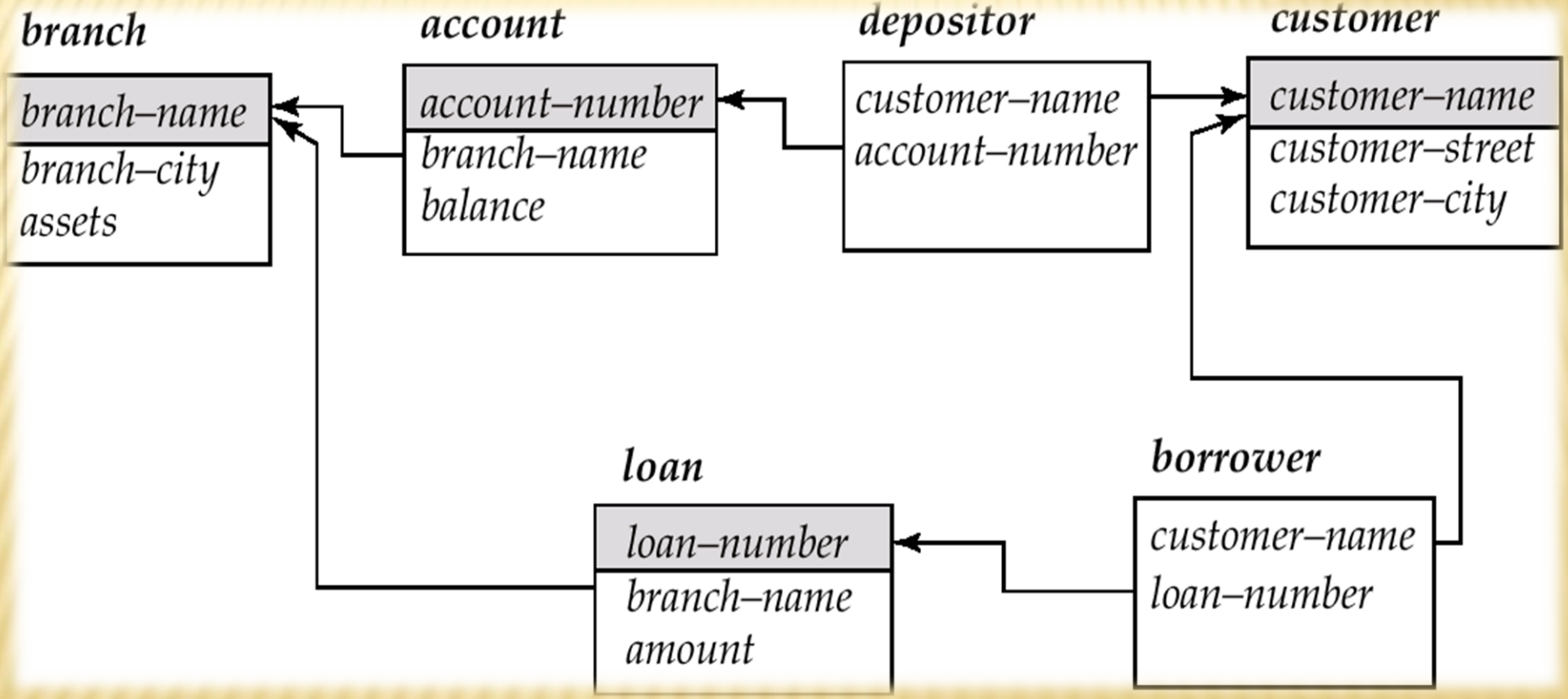
CONCEPTS OF SQL

BY
MEGHANA G RAJ

TOPICS

- Basic Structure
- Set Operations
- Aggregate Functions
- Null Values

SCHEMA USED IN EXAMPLES



BASIC STRUCTURE

- A typical SQL query has the form:

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where P

  - A_is represent attributes
  - r_is represent relations
  - P is a predicate(condition)
  - **The result of an SQL query is a relation**.

THE SELECT CLAUSE(1/3)

- The **select** clause list the attributes desired in the result of a query
- E.g. find the names of all branches in the *loan* relation

  select *branch_name*
  from *loan*
- SQL does not permit the '-' character in names,
- NOTE: SQL names are case insensitive, i.e. you can use capital or small letters.

THE SELECT CLAUSE (2/3)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after **select.**
- Find the names of all branches in the *loan* relations, and remove duplicates

  select distinct *branch-name*
  from *loan*

- The keyword **all** specifies that duplicates not be removed.

  select all *branch-name*
  from *loan*

THE SELECT CLAUSE (3/3)

- An asterisk in the select clause denotes "all attributes"

  select *
  from *loan*

- The **select** clause can contain arithmetic expressions involving the operation, +, –, *, and /, and operating on constants or attributes of tuples.
- The query:

  select *loan-number, branch-name, amount* * 100
  from *loan*

  would return a relation which is the same as the *loan* relations, except that the attribute *amount* is multiplied by 100.

THE WHERE CLAUSE(1/2)

- The **where** clause specifies conditions that the result must satisfy.
- To find all loan number for loans made at the Perryridge branch with loan amounts greater than $1200.

  select *loan-number*
  from *loan*
  where *branch-name* = 'Perryridge' **and** *amount* > 1200
- Comparison results can be combined using the logical connectives **and, or,** and **not.**
- Comparisons can be applied to results of arithmetic expressions

THE WHERE CLAUSE (2/2)

- SQL includes a **between** comparison operator
- E.g. Find the loan number of those loans with loan amounts between $90,000 and $100,000 (that is, ≥$90,000 and ≤$100,000)

select *loan-number*
from *loan*
where *amount* **between** 90000 **and** 100000

THE FROM CLAUSE

- The **from** clause lists the relations involved in the query
- Find the Cartesian product *borrower x loan*

  select ∗
  from *borrower, loan*
- Find the name, loan number and loan amount of all customers
  having a loan at the Perryridge branch.
- **select** *customer-name, borrower.loan-number, amount*
  from *borrower, loan*
  where *borrower.loan-number = loan.loan-number* **and**
  branch-name = ‘Perryridge

THE RENAME OPERATION

- The SQL allows renaming relations and attributes using the **as** clause:

  old-name **as** *new-name*

- Find the name, loan number and loan amount of all customers; rename the column name *loan-number* as *loan-id.*
- **select** *customer-name, borrower.loan-number* **as** *loan-id, amount*
  from *borrower, loan*
  where *borrower.loan-number = loan.loan-number*

TUPLE VARIABLES

- Tuple variables are defined in the **from** clause via the use of the **as** clause.
- Find the customer names and their loan numbers for all customers having a loan at some branch.
- **select** *customer-name, T.loan-number, S.amount*
  from *borrower* **as** *T, loan* **as** *S*
  where *T.loan-number = S.loan-number*
- Find the names of all branches that have greater assets than
  some branch located in Brooklyn
- **select distinct** *T.branch-name*
  from *branch* **as** *T, branch* **as** *S*
  where *T.assets > S.assets* **and** *S.branch-city* = 'Brooklyn'

STRING OPERATIONS

- SQL includes a string-matching operator for comparisons on character strings. Patterns are described using two special characters:
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.
- Find the names of all customers whose street includes the substring "Main".

```
select customer-name
from customer
where customer-street like '%Main%'
```

- Match the name "Main%"

```
like 'Main\%' escape '\'
```

- SQL supports a variety of string operations such as
  - concatenation (using "||")
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

ORDERING THE DISPLAY OF TUPLES

- List in alphabetic order the names of all customers having a loan in Perryridge branch

  select distinct *customer-name*
  from *borrower, loan*
  where *borrower loan-number = loan.loan-number* **and**
  branch-name = 'Perryridge'
  order by *customer-name*

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - E.g. **order by** *customer-name* **desc**

SET OPERATIONS

- The set operations **union, intersect,** and **except** operate on relations and correspond to the relational algebra operations $\cup$, $\cap$, $-$.
- Each of the above operations automatically eliminates duplicates; to retain all duplicates use the corresponding multiset versions **union all, intersect all** and **except all.**
- Suppose a tuple occurs *m* times in *r* and *n* times in *s*, then, it occurs:
  - *m* + *n* times in *r* **union all** *s*
  - min(*m*,*n*) times in *r* **intersect all** *s*
  - max(0, *m* – *n*) times in *r* **except all** *s*

SET OPERATIONS

- Find all customers who have a loan, an account, or both:

(select *customer-name* **from** *depositor*)
union
(select *customer-name* **from** *borrower)*

- Find all customers who have both a loan and an account

(**select** *customer-name* **from** *depositor*)
intersect
(select *customer-name* **from** *borrower)*

Find all customers who have an account but no loan

(**select** *customer-name* **from** *depositor*)
except
(select *customer-name* **from** *borrower)*

AGGREGATE FUNCTIONS(1/2)

- These functions operate on the multiset of values of a column of a relation, and return a value

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

AGGREGATE FUNCTIONS(2/2)

- Find the average account balance at the Perryridge branch

  select avg *(balance)*
  from *account*
  where *branch-name* = ‘Perryridge

- Find the number of tuples in the *customer* relation.

  select count (*)
  from *customer*

- Find the number of depositors in the bank

  select count (distinct *customer-name)*
  from *depositor*

AGGREGATE FUNCTIONS – GROUP BY

- Find the number of depositors for each branch.

  select *branch-name,* **count (distinct** *customer-name)*
  from *depositor, account*
  where *depositor.account-number = account.account-number*
  group by *branch-name*

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list

AGGREGATE FUNCTIONS – HAVING CLAUSE

- Find the names of all branches where the average account balance is more than $1,200

  select *branch-name*, **avg** *(balance)*
  from *account*
  group by *branch-name*
  having avg *(balance)* > 1200

- **Note**: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

NULL VALUES

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The predicate **is null** can be used to check for null values.
  - E.g. Find all loan number which appear in the *loan* relation with null values for *amount.*

    select *loan-number*
    from *loan*
    where *amount* **is null**
- The result of any arithmetic expression involving *null* is *null*
  - E.g. 5 + null returns null
- However, aggregate functions simply ignore nulls
  - more on this shortly

NULL VALUES AND THREE VALUED LOGIC

- Any comparison with *null* returns *unknown*
  - *E.g. 5 < null or null <> null or null = null*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*, (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: *(true* **and** *unknown) = unknown, (false* **and** *unknown) = false,*
    (unknown **and** *unknown) = unknown*
  - NOT*: (***not** *unknown) = unknown*
  - "*P* **is unknown"** evaluates to true if predicate *P* evaluates to *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

NULL VALUES AND AGGREGATES

- Total all loan amounts

  select sum (*amount)*
  from *loan*

  - Above statement ignores null amounts
  - result is null if there is no non-null amount
  - All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes.

- Thank You!