Modification of the Database

- **The content of the database may be modified using the following operations:**
  - **Deletion**
  - **Insertion**
  - **Updating**

- **All these operations are expressed using the assignment operator.**

- **Deletion**

  • **A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.**

  • **Can delete only whole tuples; cannot delete values on only particular attributes. A deletion is expressed in relational algebra by:**

$$r \leftarrow r - E$$

where *r* is a relation and *E* is a relational algebra query.

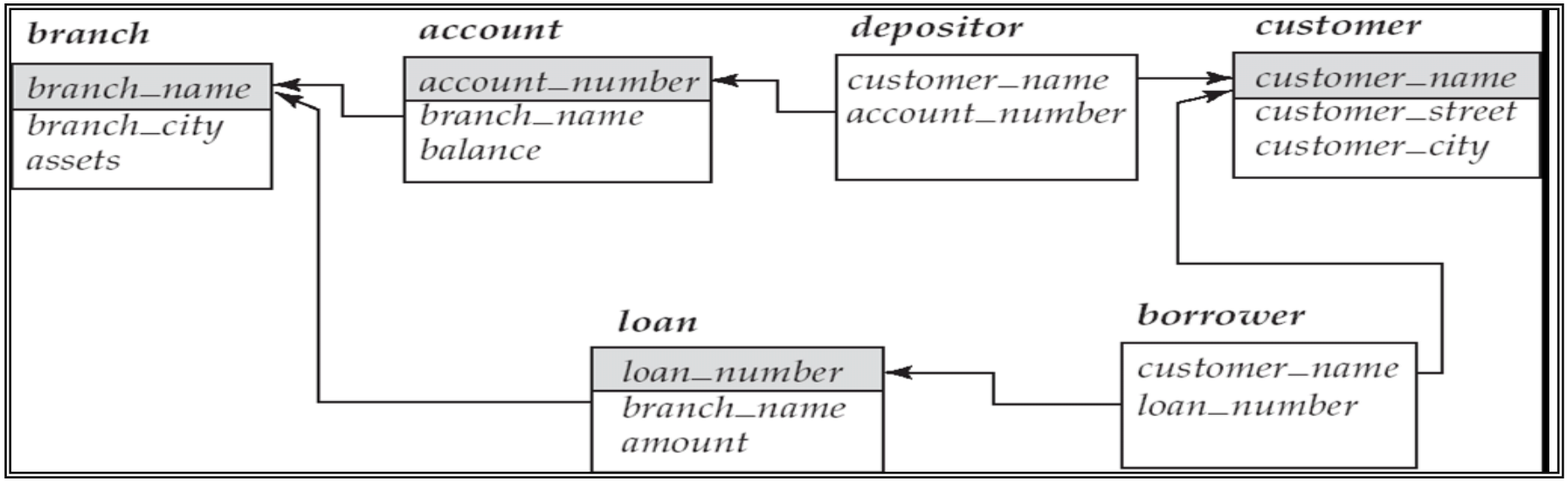
Deletion …contd.

- Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch\_name = \text{“Perryridge”}} (account)$$

- **Delete all loan records with amount in the range of 0 to 50**

- **Delete all accounts at branches located in Needham.**



Deletion …contd.

- Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch\_name = \text{“Perryridge”}} (account)$$

- **Delete all loan records with amount in the range of 0 to 50**

$$loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50} (loan)$$

- **Delete all accounts at branches located in Needham.**

$$r_1 \leftarrow \sigma_{branch\_city = \text{“Needham”}} (account \bowtie branch)$$
$$r_2 \leftarrow \prod_{account\_number,\ branch\_name,\ balance} (r_1)$$
$$r_3 \leftarrow \prod_{customer\_name,\ account\_number} (r_2 \bowtie depositor)$$
$$account \leftarrow account - r_2$$
$$depositor \leftarrow depositor - r_3$$

Insertion

- To insert data into a relation, we either:

  -specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted in relational algebra, an insertion is expressed by:

$$r \leftarrow r \cup E$$

where r is a relation and E is a relational algebra expression.

- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.

- Example:

- Insert information in the database specifying that Smith has Rs1200 in account A-973 at the Perryridge branch.

- Provide as a gift for all loan customers in the Perryridge branch, a Rs200 savings account. Let the loan number serve as the account number for the new savings account.

- **Insert information in the database specifying that Smith has Rs1200 in account A-973 at the Perryridge branch.**

$$account \leftarrow account \cup \{(\text{“A-973”}, \text{“Perryridge”}, 1200)\}$$

$$depositor \leftarrow depositor \cup \{(\text{“Smith”}, \text{“A-973”})\}$$

- **Provide as a gift for all loan customers in the Perryridge branch, a Rs200 savings account. Let the loan number serve as the account number for the new savings account.**

$$r_1 \leftarrow (\sigma_{branch\_name = \text{“Perryridge”}}(borrower \quad loan))$$

$$account \leftarrow account \cup \prod_{loan\_number,\ branch\_name,\ 200}(r_1)$$

$$depositor \leftarrow depositor \cup \prod_{customer\_name,\ loan\_number}(r_1)$$

Updating

- **A mechanism to change a value in a tuple without charging *all* values in the tuple**

- **Use the generalized projection operator to do this task**

$$r \leftarrow \prod_{F_1, F_2, \ldots, F_l} (r)$$

- **Each F_i is either**

  - **The I^{th} attribute of *r*, if the I^{th} attribute is not updated, or,**

  - **If the attribute is to be updated F_i is an expression, involving only constants and the attributes of *r*, which gives the new value for the attribute**

Update…contd.

- **Make interest payments by increasing all balances by 5 percent.**

$$account \leftarrow \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (account)$$

- **Pay all accounts with balances over Rs 10,000, 6 percent interest and**
- **pay all others 5 percent**

$$account \leftarrow \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.06}\ (\sigma_{BAL\ >\ 10000}\ (account\ ))$$
$$\cup\ \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (\sigma_{BAL\ \leq\ 10000}\ (account))$$

Structured Query Language (SQL)

History

- **IBM SEQUEL language developed as part of System R project at the IBM San Jose Research Laboratory**
- **Renamed Structured Query Language (SQL)**

- **ANSI and ISO standard SQL:**
  - **SQL-86**
  - **SQL-89**
  - **SQL-92**
  - **SQL:1999 (language name became Y2K compliant)**
  - **8.0 for SQL Server 2000.**
  - **SQL:2003**
  - **9.0 for SQL Server 2005.**
  - **10.0 for SQL Server 2008.**
  - **10.5 for SQL Server 2008 R2.**
  - **11.0 for SQL Server 2012.**
  - **12.0 for SQL Server 2014**
  - **13.0 for SQL Server 2016**
  - **14 for SQL Server 2017**
  - **15 for SQL Server 2019**
  - **16 for SQL Server 2022**
- **Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.**

Data Definition Language

- **Allows the specification of:**
  - **The schema for each relation including attribute types**
  - **Integrity constraints**
  - **Authorization information for each relation.**
- **Non-standard SQL extensions also allow specification of:**
  - **The set of indices to be maintained for each relations.**
  - **The physical storage structure of each relation on disk.**

Create Table Construct

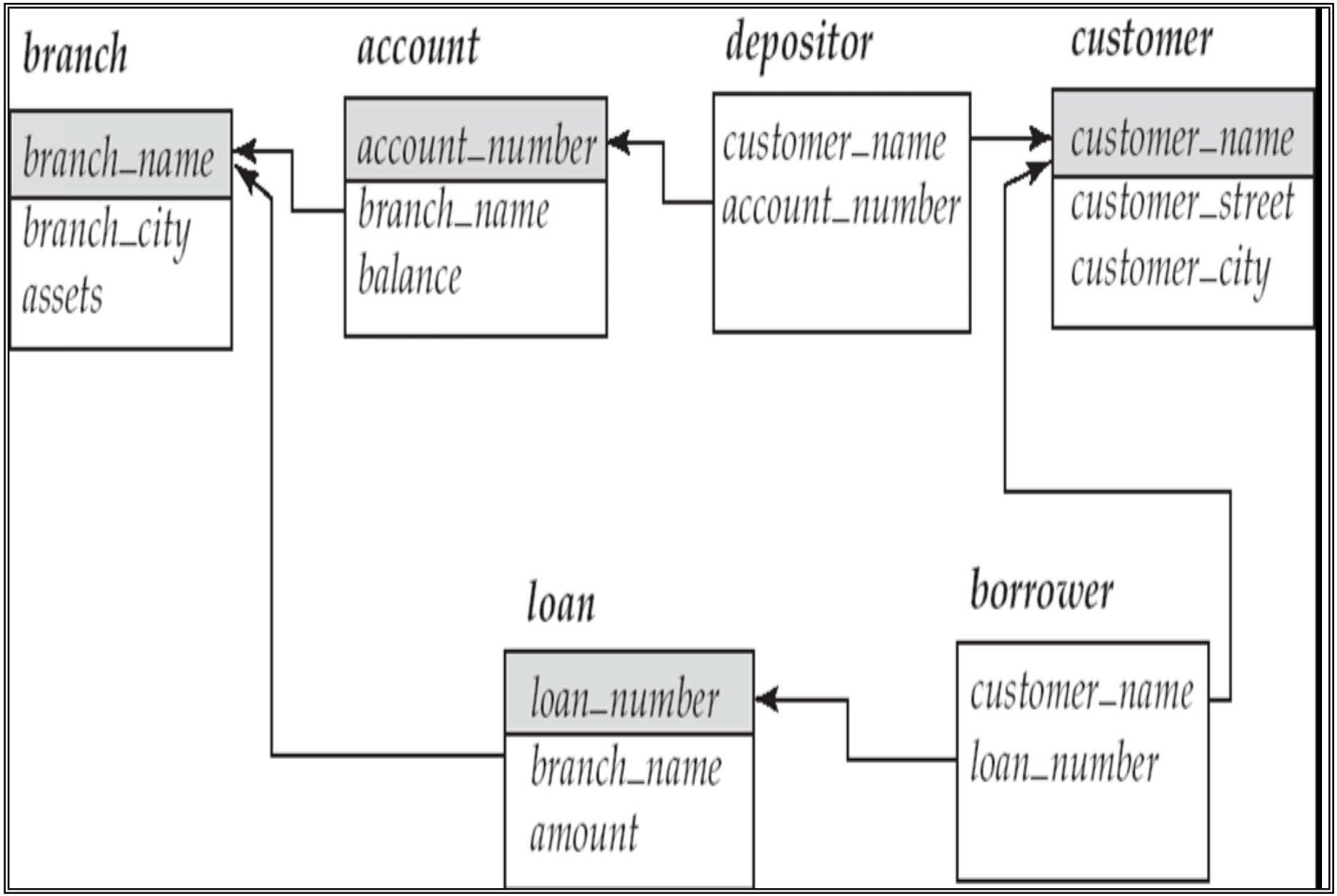
- **An SQL relation is defined using the create table command:**

  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 attribute A_i**

- **Example:**

  create table *branch*
  (*branch_name* char(15),
  ***branch_city* char(30),**
  ***assets* integer)**


branch
branch_name
branch_city
assets
account
account_number
branch_name
balance
depositor
customer_name
account_number
customer
customer_name
customer_street
customer_city
loan
loan_number
branch_name
amount
borrower
customer_name
loan_number

Referential Integrity in SQL

create table *account*
(*account_number* **char**(10)**,**
branch_name **char**(15),
balance **integer**,
primary key (*account_number),*
foreign key (*branch_name*) **references** *branch*)

create table *depositor*
(*customer_name* **char**(20)**,**
account_number **char**(10)**,**
primary key (*customer_name*, *account_number),*
foreign key (*account_number*) **references** *account*,
foreign key (*customer_name*) **references** *customer*)

create table customer
(customer_name char(20),
customer_street char(30),
customer_city char(30),
primary key (customer_name))

create table branch
(branch_name char(15),
branch_city char(30),
assets numeric(12,2),
primary key (branch_name))

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.

Integrity Constraints on Tables

- **not null**
- **primary key** (A_1, ..., A_n)

- Example: Declare *branch name* as the primary key for *branch*

.

create table *branch*
(*branch_name* char(15)**,**
branch_city char(30) **not null**,
assets integer,
primary key (*branch_name*))

primary key declaration on an attribute automatically ensures **not null** in SQL-92 onwards, needs to be explicitly stated in SQL-89

Basic Insertion and Deletion of Tuples

- Newly created table is empty

- Add a new tuple to *account*

  insert into *account*
  values ('A-9732', 'Perryridge', 1200)

  – Insertion fails if any integrity constraint is violated

- Delete *all* tuples from *account*

  delete from *account*

Drop and Alter Table Constructs

- The **drop table** command deletes all information about the dropped relation from the database.

  drop table

- The **alter table** command is used to add attributes to an existing relation:

  alter table *r* **add** *A D*

  where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*.

  - All tuples in the relation are assigned *null* as the value for the new attribute.

- The **alter table** command can also be used to drop attributes of a relation:

  alter table *r* **drop** *A*

  where *A* is the name of an attribute of relation *r*

  - Dropping of attributes not supported by many databases

Basic Query Structure

- **General form of SQL query :**

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

  - A_i represents an attribute
  - R_i represents a relation
  - P is a predicate.

- This query is equivalent to the relational algebra expression.

$$\prod_{A_1, A_2, \ldots, A_n} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m))$$

- The result of an SQL query is a relation.

The select Clause

- The select clause list the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra

- Example: Find the names of all branches in the *loan* relation:

```
select branch_name
from loan
```

- In the relational algebra, the query would be:

$$\Pi_{branch\_name}(loan)$$

- 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 ….Contd.

- **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.**

- **E.g.:**

```
select loan_number, branch_name, amount *100
from loan
```

The where Clause

- **The where clause specifies conditions that the result must satisfy**
  - **Corresponds to the selection predicate of the relational algebra.**

- **To find all loan number for loans made at the Perryridge branch with loan amounts greater than Rs1200.**

```
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.**

The from Clause

- **The from clause lists the relations involved in the query**
  - **Corresponds to the Cartesian product operation of the relational algebra.**

- **Find the Cartesian product of *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'
```

Rename Operator

SQL allows renaming relations and attributes using the as clause:

old-name* as *new-name

Example: 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 and amount 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
```

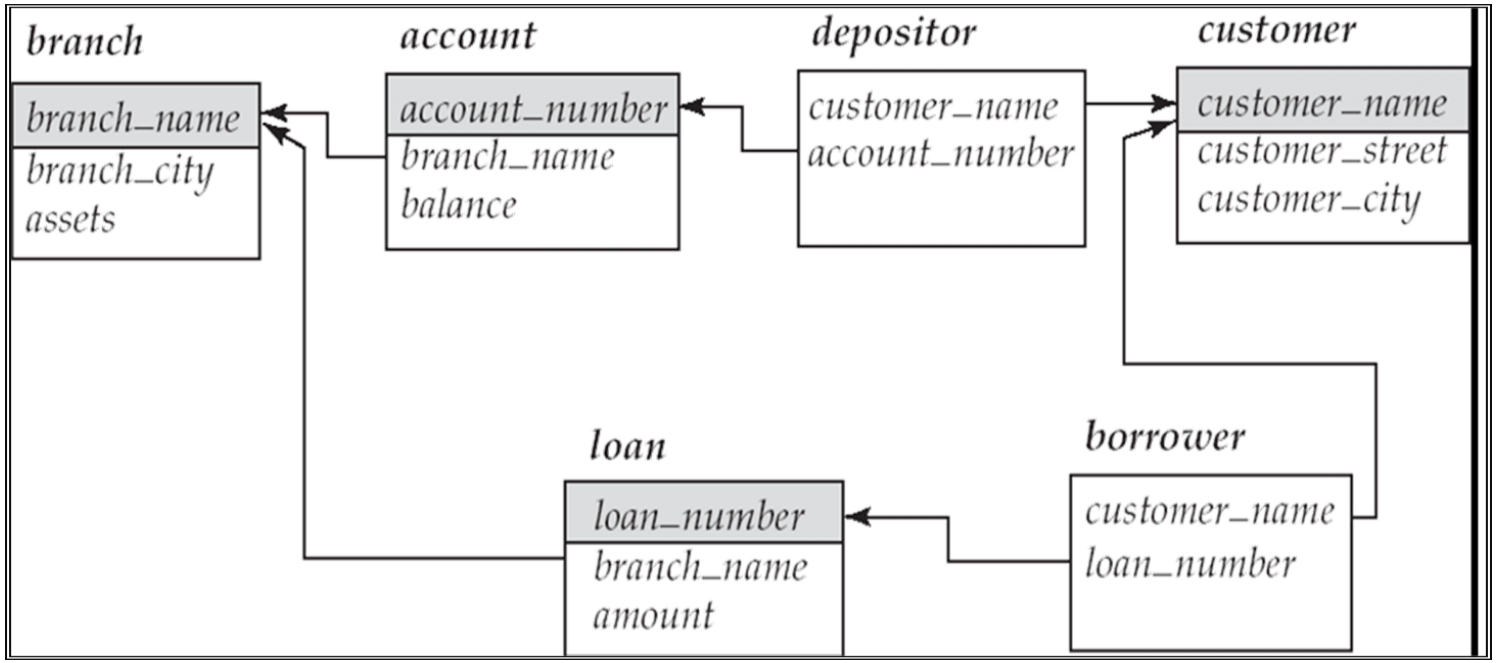
Find the names of all branches that have greater assets than some branch located in Brooklyn.

```
select distinct T.branch_name
from branch as T, branch as S
where T.assets > S.assets and S.branch_city = 'Brooklyn'
```

Keyword **as** is optional and may be omitted

borrower **as** *T* ≡ *borrower T*

Some database such as Oracle *require* **as** to be omitted


branch
branch_name
branch_city
assets
account
account_number
branch_name
balance
depositor
customer_name
account_number
customer
customer_name
customer_street
customer_city
loan
loan_number
branch_name
amount
borrower
customer_name
loan_number

String Operations

- **SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that 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.

String Operations

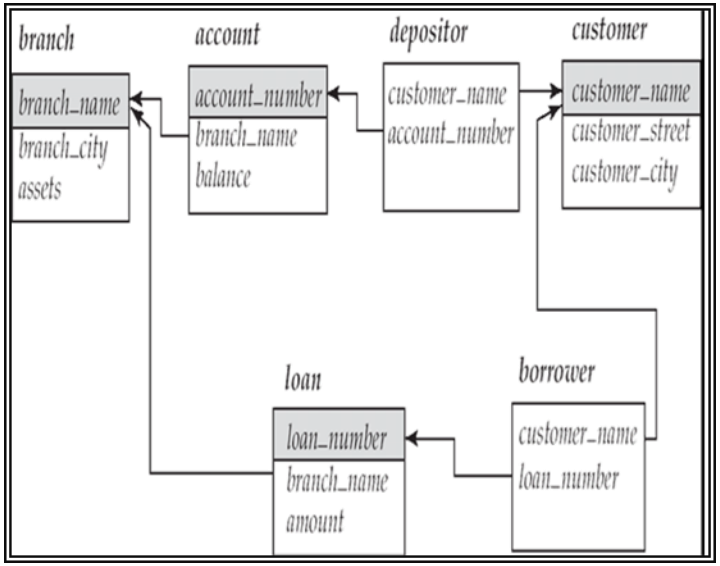
- **List in alphabetic order the names of all customers having a loan in Perryridge branch**

```
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.

Example:

```
select distinct customer_name
    from    borrower, loan
where borrower loan_number = loan.loan_number
      and branch_name = 'Perryridge'
    order by customer_name decs
```



Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result.

- **Multiset** versions of some of the relational algebra operators – given multiset relations r_1 and r_2:

Duplicates …Contd.

- SQL duplicate semantics:

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

  is equivalent to the *multiset* version of the expression:

$$\prod_{A_1, A_2, \ldots, A_n} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m))$$

Set Operations

- The set operations **union**, **intersect**, and **except** operate on relations and correspond to the relational algebra operations

- 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:**

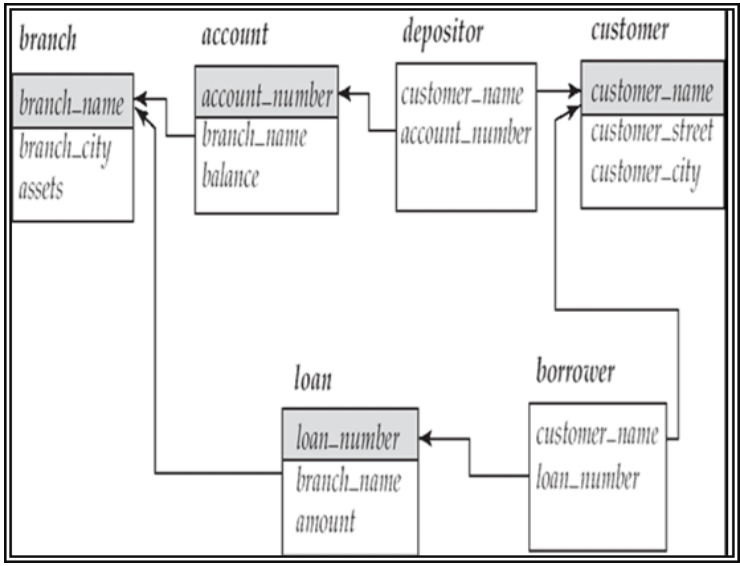
  (select customer_name **from** depositor)
  union
  (select customer_name **from** borrower)

- **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

- 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

- **Find the average account balance at the Perryridge branch.**

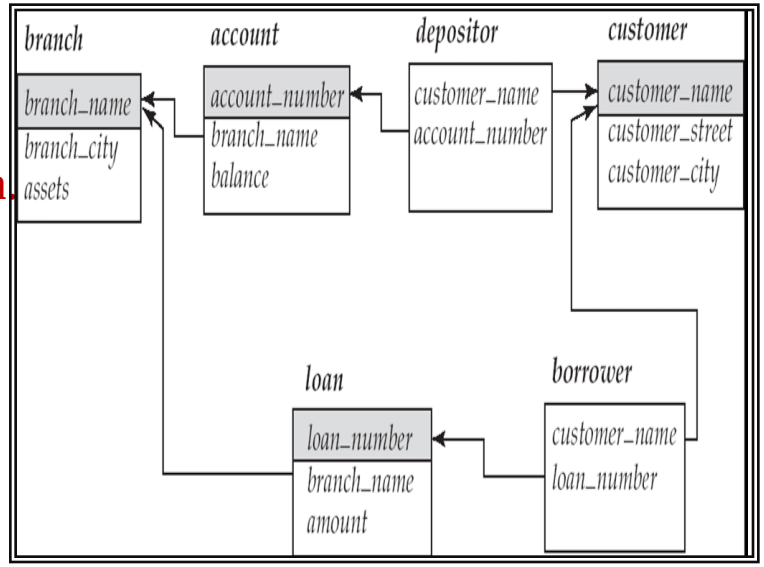
```
select avg (balance)
        from account
        where branch_name = 'Perryridge'
```

- **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
```

Aggregate Functions – Having Clause

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

```
select branch_name, avg (balance)
        from account
        group by branch_name
        having avg (balance) > 1200
```