Intro to DB

CHAPTER 3
SQL

Chapter 3: SQL

- Data Definition
- Basic Structure of SQL Queries
- Set Operations
- Aggregate Functions
- Null Values
- Nested Subqueries
- Complex Queries
- Views
- Modification of the Database
- Joined Relations

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!)
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system.

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:

  create table *branch*
  (*branch-name* **char**(15),
  branch-city **char**(30),
  assets **integer**)

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.

Integrity Constraints in Create Table

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

Declare branch_name as the primary key for branch and ensure branch_city is not null.

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

Drop and *Alter Table* Constructs

- **drop table:** deletes all information about the dropped relation from the database.
- **alter table**: used to add or drop attributes to an existing relation

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

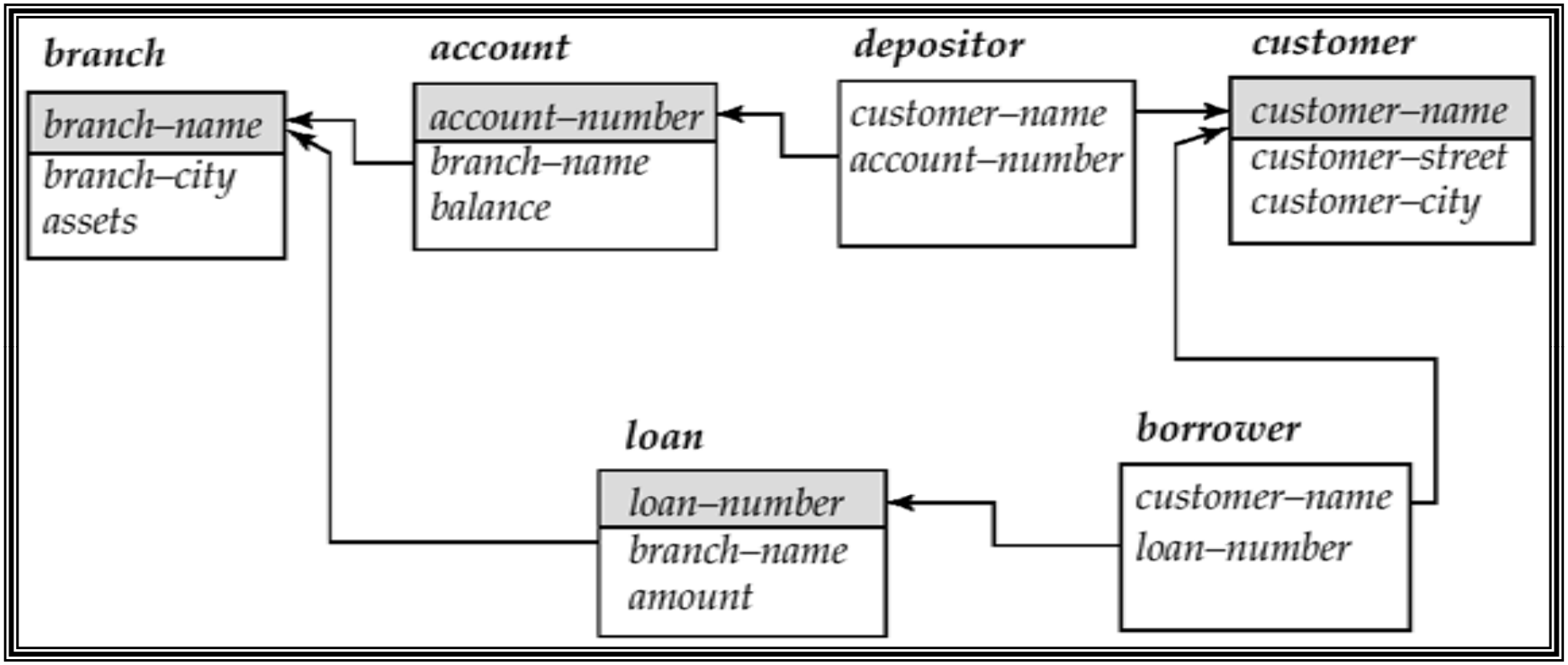
  where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*.
- *null* is assigned to the new attribute for each tuple

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

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

  (dropping of attributes is not supported by many databases)

Schema Used in Examples



Basic Structure of SQL Queries

- SQL is based on set and relational operations with certain modifications and enhancements
- 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.
- This query is equivalent to the relational algebra expression.

$$\Pi_{A1, A2, ..., An}(\sigma_P (r_1 \times r_2 \times ... \times r_m))$$

- The result of an SQL query is a relation.

The *select* Clause

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

  select *branch-name*
  from *loan*

- Corresponds to the relational algebra query (not exactly same)

  $\Pi_{branch\text{-}name}(loan)$

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

  select *
  from *loan*

- NOTE:
  - SQL does not permit the '-' character in names (use '_' in a real implementation).
  - SQL names are case insensitive.

The *select* Clause (cont.)

- 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 (cont.)

- 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

- Corresponds to the selection predicate of the relational algebra.
- Predicate involving attributes of the relations that appear in the **from** clause.

  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 conditions: >, <, =, <=, >=, !=
- Logical connectives: **and, or, not**
- Comparisons can be applied to results of arithmetic expressions.

The *from* Clause

- Corresponds to the Cartesian product operation of the relational algebra.
  - Lists the relations to be scanned in the evaluation of the expression.
- *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

- Defined in the **from** clause by use of **as**.
- 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'

- '**as**' is optional

String Operations

- For comparisons on character strings
- Patterns are described using special characters:
  - percent (%): matches any substring.
  - underscore (_). 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%'
- Escape character to specify % and \ within string

  like 'Main\%'
- A variety of string operations such as
  - concatenation (using "||")
  - case conversion, string length, 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*

- **desc** for descending order or **asc** for ascending order (default)
  - E.g. **order by** *customer-name* **desc**
  - E.g. **order by** *customer-name* **desc**, loan-number **asc**

Set Operations

- The set operations: **union, intersect, except** correspond to the relational algebra operations $\cup$, $\cap$, $-$.
- Each set operation automatically eliminates duplicates
- To retain all duplicates use 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 (cont.)

- 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

- 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 (cont.)

- 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*

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

Null Values

- Tuples may have a null value (*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.

  Find all loan numbers which appear in the *loan* relation with null values for *amount*.

  select *loan-number*
  from *loan*
  where *amount* **is null** (why not '*amount*=**null**')

- The result of any arithmetic expression involving *null* is *null*
  - E.g. 5 + null returns null
- However, aggregate functions simply ignore nulls

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

Nested Subqueries

- A subquery is a **select-from-where** expression that is nested within another query.
- Common use of subqueries:

  perform tests for set membership, set comparisons, and set cardinality.

Example Query

- Find all customers who have both an account and a loan at the bank.

  select distinct *customer-name*
  from *borrower*
  where *customer-name* **in (select** *customer-name*
  from depositor)

- Find all customers who have a loan at the bank but do not have an account at the bank

  select distinct *customer-name*
  from *borrower*
  where *customer-name* **not in (select** *customer-name*
  from *depositor)*

Modification of the Database – Deletion

- *Delete all account records at the Perryridge branch.*

```
delete from account
where branch-name = 'Perryridge'
```

- *Delete all accounts at every branch located in Needham city.*

```
delete from account
where branch-name in (select branch-name
                      from branch
                      where branch-city = 'Needham')
delete from depositor
where account-number in
                (select account-number
                 from branch, account
                 where branch-city = 'Needham'
                 and branch.branch-name = account.branch-name)
```

Modification of the Database – Deletion

- *Delete the record of all accounts with balances below the average at the bank.*

delete from *account*
where *balance* < (**select avg** *(balance)*
from *account*)

  - Problem: as we delete tuples from *deposit,* the average balance changes
  - Solution used in SQL:
    1. First, compute **avg** balance and find all tuples to delete
    2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

Modification of the Database – Insertion

- Add a new tuple to *account*

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

  or equivalently

  insert into *account (branch-name, balance, account-number)*
  values ('Perryridge', 1200, 'A-9732')

- Add a new tuple to *account* with *balance* set to null

  insert into *account*
  values ('A-777','Perryridge', *null*)

Modification of the Database – Insertion

- *Provide as a gift for all loan customers of the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account*

  insert into *account*
  select *loan-number, branch-name,* 200
  from *loan*
  where *branch-name* = 'Perryridge'
  insert into *depositor*
  select *customer-name, loan-number*
  from *loan, borrower*
  where branch-name = 'Perryridge'
  and *loan.account-number* = *borrower.account-number*

- The select from where statement is fully evaluated before any of its results are inserted into the relation

  Otherwise, queries like
  insert into *table*1 **select** * **from** *table*1
  would cause problems

Modification of the Database – Update

Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.

- Write two **update** statements:

update *account*
set *balance* = *balance* * 1.06
where *balance* > 10000

update *account*
set *balance* = *balance* * 1.05
where *balance* ≤ 10000

- The order is important

END OF CHAPTER 3