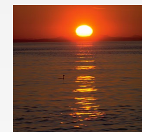
Lecture 6: SQL
Chapter 3

Database System Concepts, 5th Ed.

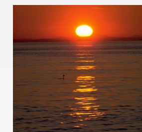
©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use



Administrative issues

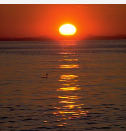
- SQL assignment
- SQL*plus documentation
  http://www.oracle.com/technology/docs/tech/sql_plus/index.html
- SQL documentation
  See SQL reference in:
  http://www.oracle.com/pls/db102/homepage
- Note – for drawing ER diagrams
  DiaCze (Windows)
  http://www.cze.cz/downloads.php
  Kivio (Linux, Mac)
  http://www.koffice.org/download/
  Dia (Linux, Windows)
  http://live.gnome.org/Dia/Download

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use



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.
- More are covered in Chapter 4.



©Silberschatz, Korth and Sudarshan

Create Table Construct

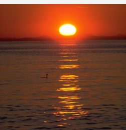
- An SQL relation is defined using the **create table** command:

  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) not null,
    branch_city     char(30),
    assets          integer)
```



Integrity Constraints in Create Table

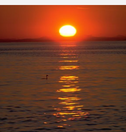
- **not null**
- **primary key** (A_1, ..., A_n)

Example: Declare *branch_name* as the primary key for *branch*

.

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

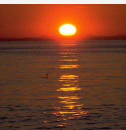
- The **drop table** command deletes all information about the dropped relation from the database.
- The **alter table** command is used to add attributes to an existing relation:

  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

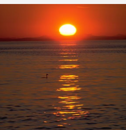
- SQL is based on set and relational operations with certain modifications and enhancements
- A typical SQL query has the form:

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

  - A_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

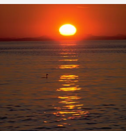
- The **select** clause list the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all branches in the *loan* relation:

  select *branch_name*
  from *loan*

- In the relational algebra, the query would be:

  $$\Pi_{branch_name}(loan)$$

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g. *Branch_Name* ≡ *BRANCH_NAME* ≡ *branch_name*
  - Some people use upper case wherever we use bold font.



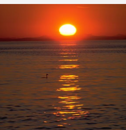
The select Clause (Cont.)

- 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.)

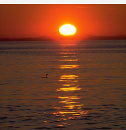
- An asterisk in the select clause denotes “all attributes”

  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*

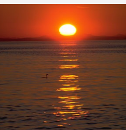
  would return a relation that is the same as the *loan* relation, except that the value of the attribute *amount* is multiplied by 100.



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 $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.

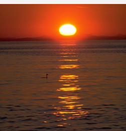


©Silberschatz, Korth and Sudarshan

The where Clause (Cont.)

- SQL includes a **between** comparison operator
- Example: 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
  - Corresponds to the Cartesian product operation of the relational algebra.
- 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'
```

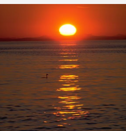
The Rename Operation

- 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*



©Silberschatz, Korth and Sudarshan

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.

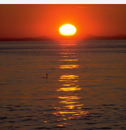
> **select** *customer_name, T.loan_number, S.amount*
> **from** *borrower* **as** *T, loan* **as** *S*
> **where** *T.loan_number* = *S.loan_number*

- 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'

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

> *borrower* **as** *T* ≡ *borrower T*



String Operations

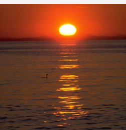
- SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.
- Find the names of all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

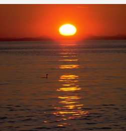


Ordering the Display of Tuples

- 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.
  - Example: **order by** *customer_name* **desc**



Set Operations

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

  (select *customer_name* **from** *depositor*)
  union [or union all]
  (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.

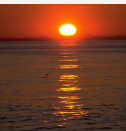
  (**select** *customer_name* **from** *depositor*)
  except
  (select *customer_name* **from** *borrower)*

Aggregate Functions – 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



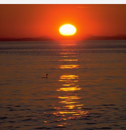
©Silberschatz, Korth and Sudarshan

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

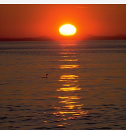


Views

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know a customer's name, loan number and branch name, but has no need to see the loan amount. This person should see a relation described, in SQL, by

  (**select** *customer_name, borrower.loan_number, branch_name*
  from *borrower, loan*
  where *borrower.loan_number* = *loan.loan_number*)

- A **view** provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**.

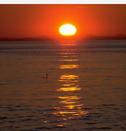


View Definition

- A view is defined using the **create view** statement which has the form

  create view *v* **as** < query expression >

  where <query expression> is any legal SQL expression. The view name is represented by *v.*
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- When a view is created, the query expression is stored in the database; the expression is substituted into queries using the view.



Example Queries

- A view consisting of branches and their customers

```
create view all_customer as
        (select branch_name, customer_name
         from depositor, account
        where depositor.account_number =
                    account.account_number )
        union
        (select branch_name, customer_name
        from borrower, loan
        where borrower.loan_number = loan.loan_number )
```

- Find all customers of the Perryridge branch

```
select customer_name
        from all_customer
        where branch_name = 'Perryridge'
```