CSE 132B

SQL as Query Language (Part I)

Some slides are based or modified from originals by
Elmasri and Navathe,
Fundamentals of Database Systems, 4th Edition

and
Database System Concepts, McGraw Hill 5th Edition

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. $\Pi_{A_1,A_2,\ldots,A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$
- 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
  - projection operation of 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:

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

The select Clause (Cont.)

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

  select distinct *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 /, operating on constants or attributes.
- The query:

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

  would return a relation similar to the *loan* relation, except that values for *amounts* are multiplied by 100.

The where Clause

- The **where** clause specifies conditions that the result must satisfy
  - Relational algebra's selection predicate.
- 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

The where Clause (Cont.)

- Comparison results can be combined using the logical connectives **and, or,** and **not.**
- Comparisons can be applied to results of arithmetic expressions.
- 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.
- Ex. *borrower X loan*

  select ∗
  from *borrower*, *loan*

  No where clause!

The from Clause (Cont.)

- Ex. 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 / Aliases

- Tuple variables are defined in the **from** clause via the use of the **as** clause.
- E.g. 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*

Tuple Variables / Aliases (cont.)

- 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 pattern matching operator for comparisons on character strings.
- The operator “like” uses patterns that are described using two special characters:
  - percent % or *: matches any substring.
  - underscore _ or ?: matches any character.
- *E.g.* Find the names of all customers whose street includes the substring “Main”.

```
select  customer_name
from    customer
where customer_street like ‘%Main%’
```

String Operations

- Streets that match the name “Main%”
  - % (or *) are part of the substring

    … **like** ‘Main\%′ **escape** ‘\′
- E.g. Any street name with exactly 5 characters

  … **like** ‘_ _ _ _ _′
- 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.

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**.

Set Operations

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

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

- Find all customers with both a loan and an account:

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

- Find all customers with an account but no loan:

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

Null Values

The predicate **is null** is used to check for null values.

- Example: Find all loan number which appear in the *loan* relation with null values for *amount*.

  select *loan_number*
  from *loan*
  where *amount* **is null**

- There is also a **is not null** option.

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A subquery is a **select-from-where** expression that is nested within another query.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

Nested Subqueries Examples

- 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.

```
select distinct customer_name
        from borrower
        where customer_name not in (select customer_name
                                    from depositor )
```

Nested Subqueries Examples

- Find all customers who have both an account and a loan at the Perryridge branch

  Note: This query can be written in a much simpler manner.
  The formulation below is simply to illustrate SQL features.

```
select distinct customer_name
        from borrower, loan
        where borrower.loan_number = loan.loan_number
        and branch_name = 'Perryridge'
         and (branch_name, customer_name ) in
                (select branch_name, customer_name
                  from depositor, account
                  where depositor.account_number = account.account_number )
```

Set comparison: the Some clause

- Find 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'
```

- Same query using > **some** clause

```
select branch_name
        from branch
        where assets > some
                (select assets
                 from branch
                 where branch_city = `Brooklyn')
```

Definition of Some Clause

- F <comp> **some** $r \Leftrightarrow \exists\, t \in r$ such that (F <comp> t)
  Where <comp> can be: $<,\ \leq,\ >,\ =,\ \neq$

(5 < **some** | 0 / 5 / 6 |) = true (read: 5 < some tuple in the relation)

(5 < **some** | 0 / 5 |) = false

(5 = **some** | 0 / 5 |) = true

(5 ≠ **some** | 0 / 5 |) = true (since 0 ≠ 5)

(= **some**) ≡ **in**
However,
(≠ **some**) ≢ **not in**

Set comparison: the All clause

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

```
select branch_name
        from branch
        where assets > all
                (select assets
                from branch
                where branch_city = 'Brooklyn')
```

Definition of all Clause

- F <comp> **all** $r \Leftrightarrow \forall\ t \in\ r\ $ (F <comp> t)

(5 < **all** {0, 5, 6}) = false

(5 < **all** {6, 10}) = true

(5 = **all** {4, 5}) = false

(5 ≠ **all** {4, 6}) = true

(since 5 ≠ 4 and 5 ≠ 6)

(≠ **all**) ≡ **not in**
However,
(= **all**) ≢ **in**

Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$
- *E.g.* Find all customers that have both an account and a loan

```
select customer_name
from borrower
where exists
        (select *
        from depositor
        where depositor.customer_name = borrower.customer_name)
```

Another Exists Query

- Find all customers who have an account at all branches located in Brooklyn.

```
select distinct S.customer_name
from depositor as S
where not exists (
        (select  branch_name
        from     branch
        where    branch_city = 'Brooklyn')
        except
        (select  R.branch_name
        from     depositor as T, account as R
        where    T.account_number = R.account_number and
                 S.customer_name = T.customer_name ))
```

- For each customer, we need to check whether the set of all branches he has an account contains the set of all branches in Brooklyn.
- *Note that* $X - Y = \emptyset \Leftrightarrow X \subseteq Y$

Joined Relations

- **Join operations** take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
  - SQL92 style: only relations in from clause. *E.g.*

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

Joined Relations (cont.)

- **Join condition** – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

Joined Relations – Datasets for Examples

- Relation *borrower* and *loan*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

- Note: borrower information missing for L-260 and loan information missing for L-155

Joined Relations – Examples (cont.)

- *loan* **inner join** *borrower* **on**
  loan.loan_number = borrower.loan_number

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

- *loan* **left outer join** *borrower* **on**
  loan.loan_number = borrower.loan_number

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	*null*	*null*

Joined Relations – Examples (cont.)

- *loan* **natural inner join** *borrower*

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

- *loan* **natural right outer join** *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Joined Relations – Examples (cont.)

- *loan* **full outer join** *borrower* **using** (*loan_number*)

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

- Find all customers who have either an account or a loan (but not both) at the bank.

select *customer_name*
from (*depositor* **natural full outer join** *borrower*)
where *account_number* **is null or** *loan_number* **is null**