II. Structured Query Language (SQL)

Lecture Topics

- Basic concepts and operations of the relational model.
- The relational algebra.
- The SQL query language.

Basic Relational Concepts

Relating to descriptions of data

- **Attribute (column):** A name denoting a property or characteristic.
- **Relation schema (table header):** A finite set of attributes and a mapping of each attribute to a domain (defined below).

Basic Relational Concepts (cont'd)

Relating to data

- **Domain:** An "abstract data type" (i.e., a name, a set of values and a number of functions defined over the values).
- **Null value:** A special exception value (usually meaning "not known" or "not applicable").
- **Tuple:** A set of attribute/value pairs, with each attribute occurring at most once.
- **Relation (table):** A relation schema, and a finite set of tuples.
- **Relational database:** A finite set of relation names and a mapping of each relation name to a relation.

Other

- **Intention of a relation:** The associated relation schema.
- **Extension of a relation:** The associated set of tuples.

The relational model assumes no ordering of either rows or columns for any table.

Basic Rules

- **Domain constraints:** If not null, then the value associated with each attribute in a tuple must occur in the set of values associated with the domain of the attribute.
- **First normal form:** Neither tuples nor relations can be values in any domain.
- **Completeness:** Each tuple in a relation has an attribute/value pair for precisely the set of attributes in the associated relation schema.
- **Closed world:** The database “knows of” all tuples in all relations.
- **Unique rows:** No two distinct tuples in any given relation consist of the same set of attribute/value pairs.

Foundational Constraints

Relating to **functional dependencies**

- **Relation superkey:** (defined earlier) A subset of the associated relation schema for which no pair of distinct tuples in the relation will **ever** agree on the corresponding values.
- **Relation candidate key:** A minimal superkey.
- **Relation primary key:** A distinquished candidate key of the relation.

Relating to **inclusion dependencies**

- **Foreign key:** (defined earlier) Primary key of one relation appearing as attributes of another relation.

Foreign keys enable capturing more complex entity structure. Can also be used to enforce subtyping.

Rules relating to Primary and Foreign Keys

- **Entity integrity:** No component of a primary key value may be the null value, or may be updated.
- **Referential integrity:** A tuple with a non-null value for a foreign key that does not match the primary key value of a tuple in the referenced relation is not allowed.

Relational Algebra

Proposed by Codd as basic means of manipulating data in a relational database.

A procedural query language.

Fundamental operations

- reference
- selection
- projection
- cross product
- set union
- set difference
- renaming

Algebra: Set of operators mapping existing relations to new relations.

Reference

Referring to an existing relation.

Notation: Id.

Value: The relation named by Id.

Ex. “The vendors.”

Vendor

Result: (a duplication of the Vendor table)

Selection

Choosing some rows from a relation.

Our notation: R WHERE C, where R is a relation and C is a condition on individual rows of R.

More typical notation: $\sigma_C(R)$ (with "sigma")

Value: Those rows of R for which condition C is true.

Ex. "The vendors in Waterloo."

Vendor WHERE City = 'Waterloo'

Result:

Vno	Vname	City	Vbal
4	Esso	Waterloo	2.25

Selection (cont'd)

Selection condition may

- test for equality ($=$) between attributes and values, and
- invoke function calls on underlying domains.

To simplify writing queries, may

- build more complicated conditions using logical connectives AND, OR and NOT.

Ex. "Vendors that are in Waterloo or have a balance exceeding 100."

Vendor WHERE (City = 'Toronto' OR Vbal $>$ 100)

Result:

Vno	Vname	City	Vbal
1	Sears	Toronto	200.00
2	Kmart	Ottawa	671.05

Projection

Drop attributes from the result.

Our notation: $R[A_1, \ldots, A_n]$, where A_i are attributes in the relational schema of R.

More typical notation: $\pi_{\{A_1,\ldots,A_n\}}(R)$ (with "pi")

Value: R restricted to attributes in A.

Ex. "Vendor names."

Vendor[Vname]

Result:

Vname
Sears
Kmart
Esso

Because relations are sets, any resulting duplicate rows are removed.

Cross Product

Pairing all possible combinations of tuples from two relations.

Our notation: (R_1 TIMES R_2), where R_1 and R_2 are relations with disjoint relational schema.

More typical notation: ($R_1 \times R_2$)

Value: Every tuple in R_1 unioned with every tuple in R_2.

May generate a very large relation:

Number of tuples in result = (number of tuples in R_1)
$\times$ (number of tuples in R_2).

Number of values in result tuple = (number of values in an R_1 tuple)
$+$ (number of values in an R_2 tuple)

Cross Product (cont'd)

Ex.

R_1

A	B
1	x
2	y

R_2

C	D
a	s
b	t
c	u

(R_1 TIMES R_2)

A	B	C	D
1	x	a	s
1	x	b	t
1	x	c	u
2	y	a	s
2	y	b	t
2	y	c	u

Set Union

Merging two relations.

Our notation: (R_1 UNION R_2), where R_1 and R_2 are relations with equivalent relational schema.

More typical notation: ($R_1 \cup R_2$)

Value: All tuples in R_1 or in R_2 (or in both).

Ex. "Vendors that are in Waterloo or have a balance exceeding 100."

(Vendors WHERE City = 'Waterloo'
UNION
Vendors WHERE Vbal $>$ 100)

Result:

Vno	Vname	City	Vbal
1	Sears	Toronto	200.00
2	Kmart	Ottawa	671.05

Set Difference

Excluding tuples of one relation.

Our notation: (R_1 EXCEPT R_2), where R_1 and R_2 are relations with equivalent relational schema.

More typical notation: ($R_1 - R_2$)

Value: Tuples in R_1 that are not in R_2.

Ex. "Vendor numbers for vendors with no transactions."

(Vendors[Vno] EXCEPT Transactions[Vno])

Result:

Vno
1

Renaming

Renaming attributes.

Our notation: $R\{A_1 \text{ AS } B_1, \ldots, A_n \text{ AS } B_n\}$.

Value: Same as R, with attribute A_i replaced by attribute B_i.

Ex. "Vendor names for vendors with no transactions."

```
(Vendor[Vname] EXCEPT
  (Vendor{Vno AS V.Vno} TIMES Transaction)
  WHERE Vno = V.Vno
  [Vname])
```

Result:

Vname
Sears

Additional Operators

Do not increase expressive power, but make life easier.

- set intersection
- natural join ($R_1 \bowtie R_2$)
- division ($R_1 \div R_2$)
- assignment

We shall use an assignment operator.

Notation: Id := R_1 IN R_2.

Value: The value of R_2, assuming that Id is a new relation name assigned the value of R_1.

Ex. "Vendor names for vendors with no transactions."

T1 := Vendor{Vno AS V.Vno} IN
T2 := (T1 TIMES Transaction) IN
T3 := T2 WHERE Vno = V.Vno [Vname] IN
(Vendor[Vname] EXCEPT T3)

The SQL Query Language

- Expressing the algebraic operators.
- More examples of querying in SQL.
- Expressiveness and limitations.

Retrieving All Information About a Table

Ex. “The vendors.”

select $*$ **from** Vendor

Selecting Data

Ex. “The vendors in Waterloo.”

select $*$ **from** Vendor
where City = ’Waterloo’

Ex. “Vendors that are in Waterloo or have a balance exceeding 100.”

select $*$ **from** Vendor
where City = ’Toronto’ **or** Vbal $>$ 100

Projecting Columns

Ex. "The names of vendors."

select distinct Vname **from** Vendor

Note: evaluating "**select** Vname **from** Vendor" will return

Vname
Sears
Kmart
Esso
Esso

In SQL, a query returns a **multiset** of tuples; that is, the same tuple can appear more than once.

Computing a Cross Product

Ex. "All combinations of vendors and customers."

select * **from** Vendor, Customer

Ex. "All pairs of vendors."

select * **from** Vendor **as** V1, Vendor **as** V2

Equivalent to

```
T1 := Vendor{Vno AS V1.Vno, Vname AS V1.Vname} IN
T2 := T1{City AS V1.City, Vbal AS V1.Vbal} IN
T3 := Vendor{Vno AS V2.Vno, Vname AS V2.Vname} IN
T4 := T3{City AS V2.City, Vbal AS V2.Vbal} IN
(T2 TIMES T4)
```

Computing a Set Union

Ex. "Vendors that are in Waterloo or have a balance exceeding 100."

(**select** $*$ **from** Vendor **where** City = 'Waterloo')
union
(**select** $*$ **from** Vendor **where** Vbal $>$ 100)

Allowed at top level only in the SQL89 standard.

Set Difference

First defined for level 2 in the SQL2 standard.

Ex. "Vendor numbers for vendors with no transactions."

(**select** Vno **from** Vendor) **except** (**select** Vno **from** Transaction)

Can also use **exists** predicate to compute set difference.

select Vno **from** Vendor **as** V
where not exists (**select** ∗ **from** Transaction **as** T
where T.Vno = V.Vno)

Renaming Columns

Ex. "Names of vendors and customers."

(**select** Vname **as** Name **from** Vendor)
union
(**select** Cname **as** Name **from** Customer)

Ex. "Total transaction amount for Esso."

select sum(Amount) **as** "Transaction Total"
from Vendor, Transaction
where Vendor.Vname = 'Esso'
and Vendor.Vno = Transaction.Vno

More On SQL Queries

in : set membership

Ex. "Vendor names for vendor numbers 1, 2 and 3."

select Vname **from** Vendor
where Vno **in** (1,2,3)

Membership testing often useful with subqueries.

Ex. "Names of vendors with no transactions on January 16, 1994."

select Vname **from** Vendor
where Vno **not in** (**select** Vno **from** Transaction
where Tdate = 940116)

Result:

Vname
Sears
Esso
Esso

Recall that SQL does not remove duplicates automatically.

More On SQL Queries (cont'd)

Avoiding duplicates: **distinct**

select distinct Vname **from** Vendor
where Vno **not in** (**select** Vno **from** Transaction
where Tdate = 940116)

Result:

Vname
Sears
Esso

More On SQL Queries (cont'd)

Table aliases (tuple variables).

Alias: A name for referring to a table.

Ex. "Names of customers and vendors that have a common transaction."

select Vname, Cname
from Customer, Transaction, Vendor
where Transaction.Acc# = Customer.AccNum
and Transaction.Vno = Vendor.Vno

Column names appearing in several tables must be preceded by the table name.

More On SQL Queries (cont'd)

Aliases enable multiple reference to the same table.

Ex. "Account numbers of customers who have transactions with both vendors 2 and 3."

select distinct T1.Account
from Transaction **as** T1, Transaction **as** T2
where T1.Acc# = T2.Acc#
and T1.Vno = 2
and T2.Vno = 3

(∼ rename operator of relational algebra)

More On SQL Queries (cont'd)

Comparing values to members of a set.

Value op **some** *Set*: true if comparison *Value op* x is true for some member x of *Set*

Value op **all** *Set*: true if comparison *Value op* x is true for all members x of *Set*

Ex. "Accounts whose balance is greater than the balance of some vendor."

select AccNum **from** Customer
where Cbal $>$ **some** (**select** Vbal **from** Vendor)

Ex. "Accounts that have the highest balance."

select AccNum, Cbal **from** Customer
where Cbal $>=$ **all** (**select** Cbal **from** Customer)

More On SQL Queries (cont'd)

Testing for (non-)emptiness of a subquery.

exists *Sub-query*: true if value of *Sub-query* contains at least one tuple.

Ex. "Names of customers with all transactions on vendors in the same city."

```
select Cname from Customer C
where exists (
        select * from Transaction T1, Vendor V1
        where T1.Acc# = C.AccNum
        and T1.Vno = V1.Vno
        and not exists (
                select * from Transaction T2, Vendor V2
                where T2.Acc# = C.AccNum
                and T2.Vno = V2.Vno
                and V1.City <> V2.City ) )
```

More On SQL Queries (cont'd)

Ordering results.

Ex. "Names of customers living in Ontario, in alphabetical order."

select Cname **from** Customer
where Prov = 'Ont'
order by Cname

Ex. "Vendor cities, names and balances in alphabetical order of vendor names and in descending order of balances."

select City, Vname, Vbal **from** Vendor
order by Vname, Vbal **desc**

Result:

City	Vname	Vbal
Waterloo	Esso	2.25
Montreal	Esso	0.00
Ottawa	KMart	671.05
Toronto	Sears	200

More On SQL Queries (cont'd)

Additional operators.

like pattern: string pattern matching.

% matches any string (incl. empty), _ matches any single character.

Ex.

Employee

Name	Street
A. Wong	123 Elm street
B.C. Wong	1 Elm street
E.F. Wong	456 Elm street
G.H.I. Wong	456 Elm street

"Employees whose name consists of 'Wong' preceded by five characters, and who live on Elm street."

select Name **from** Employee
where Name **like** '____Wong'
and Street **like** '%Elm street'

More On SQL Queries (cont'd)

Attr **between** Value1 **and** Value2

$\equiv$ Attr >= Value1 **and** Attr <= Value2.

Ex. "Names of vendors whose balance is between $100 and $500."

select VName **from** Vendor
where VBal **between** 100 **and** 500

More On SQL Queries (cont'd)

Aggregate functions.

count(*): (number of tuples)

count([distinct] column): (number of [nonduplicate] values)

sum([distinct] expr): (sum of values)

avg([distinct] expr): (average of values)

max(expr): (largest value)

min(expr): (smallest value)

Except for **count**(*), these operate on non-null values.

More On SQL Queries (cont'd)

Grouping rows together.

group by *Columns*: partition relation into groups with a common value in *Columns*.

Ex. "The total amount of transactions for each account."

select Acc#, **sum**(Amount)
from Transaction
group by Acc#

Result:

Acc#	SUM(Amount)
101	38.25
102	16.13
103	52.12

select list with **group by** may contain only attributes that are used for grouping, or aggregate functions applied to the groups.

More On SQL Queries (cont'd)

Groups can be qualified using **having** .

Ex. "The total amount of transactions for accounts that have more than one transaction."

select Acc#, **sum**(Amount)
from Transaction
group by Acc#
having count(*) $>$ 1

Result:

Acc#	SUM(Amount)
101	38.25
103	52.12

SQL Query Syntax
(simplified)

For a *select* query

select [**distinct**] [* | *term* **as** *attr* {, *term* **as** *attr* }]
from *table* **as** *alias* {, *table* **as** *alias* }
[**where** *cond*]
[**group by** *attr* {, *attr* }]
[**having** *cond*]

For queries in general

query ::= *select*
| *query* **union** [**all**] *query*
| *query* **intersect** [**all**] *query*
| *query* **except** [**all**] *query*
| **with** T **as** (*query*) {, T **as** (*query*) } *query*

For top level queries

query
[**order by** *attr* [**asc** | **desc**] {, *attr* [**asc** | **desc**] }]

select **Query Syntax**
(less simplified)

select [**distinct**] [* | *term* **as** *attr* {, *term* **as** *attr* }]
from [*table* | (*query*)] **as** *alias* {, [*table* | (*query*)] **as** *alias* }
[**where** *cond*]
[**group by** *attr* {, *attr* }]
[**having** *cond*]

Semantics of a *select* Query

- Compute cross product of all tables in **from** clause.
- Eliminate rows not satisfying **where** condition.
- Group rows according to **group by** clause.
- Eliminate groups not satisfying **having** condition.
- Evaluate expressions in **select** target list.
- Eliminate duplicate rows if **distinct** specified.

The Power of the SQL Query Language

Can express anything in the relational algebra, and more.

- Result of a query can have duplicate tuples.
- Result of a query can be ordered.
- Can count.

There are limitations.

- No “list operations” on (sub)queries.
- Null value semantics is procedural.
- Not Turing complete (e.g. cannot express recursive queries).