Relational Commercial Languages

SQL

- Structured Query Language
- Originally called Sequel
- The standard relational database language
- Several parts of the language
  - Data definition language – Commands for defining relation schemes, deleting relations, creating indices, and modifying relation schemes
  - Interactive data manipulation language
    * Based on both relational algebra and tuple calculus
    * Derived from SQUARE
    * Includes commands to insert, delete, and modify tuples
  - Embedded data manipulation language – for use with general-purpose programming languages
  - View definition
  - Authorization – access rights and views
  - Integrity
  - Transaction control
- Basic Structure
  - Inherited from SQUARE
  - Three clauses – select, from, and where
  - **select**
    * Projection operation of relational algebra
    * List the attributes desired in the result of a query
  - **from**
    * Cartesian product operation in relational algebra
    * Lists the relations to be used to evaluate the expression
  - **where**
    * Selection predicate of the relational algebra
    * Use attributes of the relation that appear in the **from** clause
- Syntax of SQL query

  select ⟨attribute list⟩
  from ⟨relation list⟩
  where ⟨condition⟩

  - If the **where** clause is omitted, it is evaluated to be true
  - If all attributes are to be selected, the ⟨attribute list⟩ can be replaced by an asterisk *
- Example queries
  - "What are the names and attributes of subparts of part 211?"

```
select   partname, part#
from     pinfo
where    sub-part-of = 211
```

- "What parts are available at Boston or O'Hare?"

```
select   part#
from     in-stock
where    location = "Boston" or location = "O'Hare"
```

- The logical connectives **and**, **or**, and **not** can be used to combine comparisons.
- SQL does not automatically eliminate duplicates
- Duplicates can be eliminated by using the keyword **distinct** (or **unique**) after **select**
- Get rid of duplicates in the last example

```
select   distinct part#
from     in-stock
where    location = "Boston" or location = "O'Hare"
```

- Set membership
  - SQL has constructs for tests involving sets of values and sets of tuples
  - Sets can be listed explicitly or can be the result of a subquery
  - Tests can be made for membership, emptiness, inclusion, and comparison with members of a set, one at a time
  - Previous query again with set notation

```
select   distinct part#
from     in-stock
where    location in ("Boston", "O'Hare")
```

  - "What are the names of parts at JFK?"

```
select   part-name
from     pinfo
where    part# in
           (select   part#
            from     in-stock
            where    location = "JFK")
```

  - "Which location has the fewest of part 211?"

```
select   location
from     in-stock
where    quantity <= all
           (select   quantity
            from     in-stock
            where    part# = 211)
```

  - "Which location stocks all the parts that Boston does?"

```
select   location
from     in-stock
where    set(part#) contains
           (select   part#
            from     in-stock
            where    location = "Boston")
```

  - set(part#) refers to the set of all part#-values occurring with a location-value
  - Possible for a subquery to reference fields from the relation in the containing query

- If there are identical field names in the containing query and subquery, they are qualified with the relation name
- Relation name serves as the tuple variable bound to the relation
- "Which locations have at least as many of a part as are named on a 707?"

```
select  part#, location
from    in-stock
where   part# in
          (select  part#
           from    used-on
           where   p-type = "707" and nused <= quantity)
```

- Another way to solve the same query

```
select  part#, location
from    in-stock
where   quantity >= any
          (select  nused
           from    used-on
           where   p-type = "707" and in-stock.part# = used-on.part#)
```

- Tuple variables
  - "What parts are available at more than one location?"

```
select  distinct part#
from    in-stock i
where   part# in
          (select  part#
           from    in-stock
           where   location not = i.location)
```

- Predicates and Joins
  - "What are the names and quantities of parts at JFK?"

```
select  part-name, quantity
from    pinfo, in-stock
where   pinfo.part# = in-stock.part# and location = "JFK"
```

  - "What parts are subparts of the same part?"

```
select  p1.part#, p2.part#
from    pinfo p1, pinfo p2
where   p1.sub-part-of = p2.sub-part-of and
        p1.sub-part-of not = 0 and
        p1.part# = p2.part#
```

  - **between** comparison operator
    * The predicate

      balance <= 100000 **and** balance >= 90000

      can be expressed as

      balance **between** 90000 **and** 100000

    * Negation expressed as

      balance **not between** 90000 **and** 100000

  - String matching patterns are described by two special characters
    * percent (%) – Matches any substring
      "%idge%" matches any of the strings "Perryridge," "Rock Ridge," "Mianus Bridge," and "Ridgeway."
    * underscore (_) – Matches any character

- **like** operator
  - street **like** "%idge%"
- **escape** operator
  - **like** "ab\\cd%" **escape** "\" matches all strings beginning with "ab\cd"

- Set operations
  - SQL includes the operations **union**, **intersect**, and **minus**
  - "Which parts are not stocked by any location?"

```
select   part#
from     pinfo
minus
select   part#
from     in-stock
```

  - Duplicates in **union** can be retained by using **union all**

- Ordering the display of tuples
  - Order of columns taken from the order of attributes in **select** clause
  - Order of tuples controlled by **order by**-clause
    - List of attributes , each followed by **asc** or **desc**
  - "List the *in-stock* relation by decreasing quantity and increasing part number

```
select     *
from       in-stock
order by   quantity desc, part# asc
```

- Aggregate functions
  - SQL allows arithmetic expressions and provides the aggregate operators **avg**, **min**, **max**, **sum**, and **count**
  - "Which locations have at least half the number of a part used in a given aircraft?"

```
select   used-on.part#, ptype, location
from     used-on, in-stock
where    used-on.part# = in-stock.part# and nused * 0.5 <= quantity
```

  - "How many coach seats are in stock?"

```
select   sum(quantity)
from     in-stock
where    part# in
           (select   part#
            from     pinfo
            where    part-name = "coach seat")
```

  - No column names corresponding to aggregates or expressions
  - Anonymous columns referred to by an integer denoting the column position for **order by** clause
  - **group by**-clause to partition tuples before the application of an aggregate operator
  - **having**-clause used to remove some groups of tuples
  - "How many of each part are there?"

```
select     part#, sum(quantity)
from       in-stock
group by   part#
```

- "For which parts is the total in stock at least the number used by a 707, and by what amount does the total exceed the number used?"

  select *used-on*.part#, nused - **sum**(quantity)
  from *used-on*, *in-stock*
  where *used-on*.part# = *in-stock*.part# **and** ptype = "707"
  group by *used-on*.part#
  having nused - **sum**(quantity) ¿ = 0

- Set comparison – Read on your own
- Testing for empty relations – Read on your own
- Power of SQL – Read on your own
- Modifying the database
  - Deletion

    delete r
    where P

    - **delete** command works only on one relation
    - **where** clause can be expressed as a set of **select** commands
  - Insertion

    insert **into** r
    values $(a_1, a_2, \ldots, a_n)$

    - Attributes can be specified as a part of relation name to allow for different sequence in a tuple
    - **values** clause can be replaced by a **select**
    - Embedded **select** cannot reference to the same relation in which values are being inserted
  - Updates

    update r
    set $a_i = \langle$new-value$\rangle$
    where P

    - **where** clause is optional
    - $\langle$new-value$\rangle$ can be replaced with a function on attribute value
- Null values
  - Possible to assign *null*-values to any attribute
  - Special predicate **null** can be used to test for null values
  - All comparisons involving null are false by definition
  - All aggregate operations except **count** ignore tuples with null value on the argument attributes
- Views
  - Created by **create view** command

    create view v **as** $\langle$query expression$\rangle$

  - View update anamoly exists
  - Some versions restrict the updation through a view
- Data definition
  - Scheme for each relation
  - Domain of values associated with each attribute

$$\textbf{create table } r(A_1 D_1, A_2 D_2, \ldots, A_n D_n)$$

  * **create table** command also includes options to specify certain integrity constraints
  * Newly created relation is initially empty
  * Data can be loaded into the new relation by **insert**

- Relations can be removed by

$$\textbf{drop table } r$$

- Relation scheme can be modified by

$$\textbf{alter table } r \textbf{ add } AD$$

Query-by-Example

- Two-dimensional syntax
- Queries are expressed "by example"
- Closely based on domain relational calculus
- Basic structure
  - Queries expressed using *skeleton tables*
  - Skeletons contain a relation name and its attributes
  - Selected skeletons filled with "example rows"
  - Example row contains constants and "example elements"
  - Domain variables are preceded by an underscore character (_)
  - The operator **P.**, for print, is prefixed to any variable or constant to appear in the result of the query
- Simple queries
  - "Which locations stock part 211?"

in-stock	part#	location	quantity
	211	**P.**_Chicago	_25

  - If a variable is mentioned in only one place, it can be omitted
  - Another way to satisfy the last query

in-stock	part#	location	quantity
	211	**P.**	

  - Comparisons other than equality are done by prefixing a constant or variable with the comparator
  - θa represents the subset of the domain of the column equal to $\{c | c\theta a\}$
  - "Which parts have more than 50 in stock at some location?"

in-stock	part#	location	quantity
	P.		>= 50

  - QBE automatically eliminates duplicates
  - "For which parts are more than 100 used on an aircraft other than a 727?"

used-on	part#	ptype	nused
	P.	¬=727	>= 100

  - Queries are not limited to one row. Multiple rows may be used.
  - "Which parts are in stock at Boston and O'Hare?"

in-stock	part#	location	quantity
	P._100	Boston	
	_100	O'Hare	

- Using the print operator in multiple rows gives the union of the results specified by each row
- "Which parts are in stock at Boston or O'Hare?"

in-stock	part#	location	quantity
	P._100	Boston	
	P._101	O'Hare	

- The following query retrieves information on part 211.

pinfo	part#	sub-part-of	partname
	P.211	**P.**	**P.**
	P.	**P.**211	**P.**

- The print operator can be applied to the entire row by placing it at the left end of the row
- Last query can also be written as

pinfo	part#	sub-part-of	partname
P.	211		
P.		211	

- "Which locations have parts that Boston does not, and what are the parts?"

instock	part#	location	quantity
	P._100	**P.**_Chicago	
_100	¬Boston		

- Facilities to match substrings
- "Find all parts where the partname begins with 'seat'."

pinfo	part#	sub-part-of	partname
	P.		**P.**seat_belt

- Queries on several relations
  - Rows are bound to various relations with tags
  - Separate skeleton is used for each relation involved in query
  - "What are the names of parts at JFK?"

pinfo	part#	sub-part-of	partname
	_100		**P.**_bolt

instock	part#	location	quantity
	_100	JFK	

  - "Which locations have at least as many of a part as are used on a 707?"

usedon	part#	ptype	nused
	_100	707	_50

instock	part#	location	quantity
	_100	**P.**_Chicago	>= _50

- The result relation
  - values from multiple relations must be combined
  - variables in the same column of one relation are to appear in different columns
  - a column needs to be renamed

- "What are the names and quantities of parts at JFK?"

pinfo	part#	sub-part-of	partname
	_100		**P.**_bolt

instock	part#	location	quantity
	_100	JFK	_50

jfkparts	partname	quantity
P.	_bolt	_50

- The condition box
  - Used to hold constraints that cannot be specified with a skeleton
  - "Which parts are subparts of the same part?"

pinfo	part#	sub-part-of	partname
	_100	_200	
	_101	_200	

Conditions
_200 ¬= 0
_100 < _101

subparts	subpart1	subpart2
P.	_100	_101

- Ordering the display of tuples
  - Controlled by the prefixes **AO.** (ascending order) and **DO.** (descending order) in the appropriate columns
  - A number in parentheses after **AO.** or **DO.** specifies the precedence of columns (if multiple ordering specified)
  - List the *instock* relation by decreasing quantity and increasing part#

instock	part#	location	quantity
P.	**AO**(2).		**DO**(1).

- Arithmetic expressions
  - "Which locations have at least half the number of a part used on a given aircraft?"

usedon	part#	ptype	nused
	_100	_DC10	_25

instock	part#	location	quantity
	_100	_Chicago	>= 0.5 * _25

hashalf	part#	ptype	location
	_100	_DC10	_Chicago

  - "What proportion of the number of coach seats used on a 707 does each location have?"

pinfo	part#	sub-part-of	partname
	_100		coach seat

usedon	part#	ptype	nused
	_100	707	_25

instock	part#	location	quantity
	_100	_Chicago	_30

seats	location	proportion
	_Chicago	_30/_25

- Aggregate operations
  - Operations available: **CNT.**, **SUM.**, **AVG.**, **MAX.**, and **MIN.**
  - "How many of part 211 are in stock?"

instock	part#	location	quantity
	211		**P.SUM.All.**_50

  - To eliminate duplicates in a set formed by **ALL.**, the operator **UNQ.** is used
  - "How many locations have parts?"

instock	part#	location	quantity
		P.CNT.UNQ.ALL.	

  - "Which parts have the number needed on a 727 in at least one location?"

usedon	part#	ptype	nused
	_100	727	_50

instock	part#	location	quantity
	P.G._100		**MAX.ALL.**_25

Conditions
MAX.ALL._25 >= _50

  - "How many of each part are there?"

instock	part#	location	quantity
	P.G.		**P.SUM.ALL.**

  - "At how many locations is each part stocked?"

instock	part#	location	quantity
	P.G.	**P.CNT.ALL.**	

- Modifying the database – Reading Assignment

QUEL

- Query language for the Ingres database system
- QUEL $\equiv$ QUEry Language
- INGRES $\equiv$ INteractive Graphics and REtrieval System
- Based on tuple relational calculus
- Tuple variables are quantified and bound to relations

range of ⟨tuple variable⟩ **is** ⟨relation name⟩

- The attribute A of a tuple variable x is denoted as $x.A$
- The basic form of a retrieval in QUEL is

retrieve (⟨target list⟩)
where ⟨condition⟩

- Target list is a sequence of tuple variable components

- Condition corresponds to a formula in tuple calculus expression
- QUEL does not have relational algebra operators like **intersect**, **union**, and **minus**
- QUEL does not allow nested subqueries
- Simple queries
  - "What is the name of part 2116?"

    range of p **is** *pinfo*
    retrieve (p.partname)
    where p.part# = 2116

  - The comparison operators =, ! =, >, >=, <=, and < are allowed in comparisons
  - Logical connectives **and**, **or**, and **not** can be used to combine comparisons
  - Attributes for the result of a retrieval are taken from corresponding tuple variable components
  - "What are the names and quantities of parts at O'Hare?"

    range of p **is** *pinfo*
    range of i **is** *instock*
    retrieve (p.partname, i.quantity)
    where p.part# = i.part# **and** i.location = "O'Hare"

- Tuple variables
  - Attributes in the result relation can be changed by using

    newname = x.oldname

    in the target list
  - Renaming must be done if the resulting relation will have the same attribute twice
  - "Which parts are subparts of the same part?"

    range of p **is** *pinfo*
    range of $p1$ **is** *pinfo*
    retrieve (subpart1 = p.part#, subpart2 = $p1$.part#)
    where p.sub-part-og = $p1$.sub-part-of **and**
    p.sub-part-of ! = 0 **and**
    p.part# < $p1$.part#

  - Condition part of the query is optional
  - Keyword **all** can be used to represent all components of a tuple variable in the target list
  - Print the entire *usedon* relation

    range of u **is** *usedon*
    retrieve u.**all**

  - Computational expressions can appear in place of a tuple variable
  - An expression appearing in the target list must be renamed
  - "Which locations have at least half the number of a part used in a given aircraft?"

    range of u **is** *usedon*
    range of i **is** *instock*
    retrieve (u.part#, u.ptype, i.location)
    where u.part# = i.part# **and**
    u.nused * 0.5 <= i.quantity

  - "What proportion of the number of coach seats used on a 707 does each location have?"

```
range of   p is pinfo
range of   u is usedon
range of   i is instock
retrieve   (i.location, proportion = (i.quantity/u.nused))
where      p.part# = u.part# and
           p.partname = "coach seat" and
           u.part# = i.part# and
           u.ptype = "707"
```

- Assignment of the result of a retrieval into a [temporary] relation is achieved via the notation "**into** ⟨relation⟩" after **retrieve**
- Consider the following query

```
range of   u is usedon
retrieve   into used727 (u.all)
where      u.ptype = "727"
```

- Aggregate functions
  - QUEL provides the aggregate operators **count**, **min**, **max**, **avg**, and **sum**
  - Aggregate operators compute functions on groups of tuples
  - Grouping specified as part of the aggregate expression
  - "How many different parts are there?"

```
range of   p is pinfo
retrieve   (numparts = count(p.part#))
```

  - "Which part has the most in stock and which part has the least in stock at any given location?"

```
range of   i is instock
retrieve   (i.part#, i.location)
where      i.quantity = max(i.quantity) or
           i.quantity = min(i.quantity)
```

  - The aggregate operators **count**, **avg**, and **sum** have "unique" versions, distinguished by a "**u**" on the end, that eliminates duplicates before applying the operator
  - "How many locations have parts?"

```
range of   i is instock
retrieve   (numlocs = countu(i.location))
```

  - "How many of part 211 are in stock?"

```
range of   i is instock
retrieve   (total = sum(i.quantity where i.part# = 211))
```

  - The following query does not answer the question: "How many seat belts are in stock?"

```
range of   i is instock
range of   p is pinfo
retrieve   (numbelts = sum(i.quantity))
where      p.part# = i.part# and
           p.partname = "seat belt"
```

  - Grouping feature for aggregates invoked with the keyword **by** within the argument to an aggregate operator
  - "How many of each part are there?"

```
range of   i is instock
retrieve   (i.part#, total = sum(i.quantity by i.part#))
```

  - "How many seat belts are there in stock?"

range of i **is** *instock*
range of p **is** *pinfo*
retrieve (numbelts = **sum**(i.quantity **by** i.part#))
where p.part# = i.part# **and**
p.partname = "seat belt"

– "How many of each part over the maximum needed by any aircraft does each location have in stock?"

range of u **is** *usedon*
range of i **is** *instock*
retrieve (u.part#, i.location, overstock = (i.quantity - **max**(u.nused **by** u.part#)))
where u.part# = i.part# **and**
(i.quantity - **max**(u.nused **by** u.part#)) > 0

- Modifying the database – Reading assignment