DSA5104
Principles of Data Management and Retrieval

Lecture 8: Complex Data Types

Recap

- Hadoop - Big data processing framework
  - Hadoop distributed file system (HDFS)
    - Architecture - NameNode / DataNode
  - MapReduce
    - Map / Shuffle & Sort / Reduce
    - Example - word count / maximum temperature
    - Use MapReduce to implement join
  - Hadoop ecosystem
- Spark
  - Spark vs MapReduce
  - Spark stack
  - RDD - resilient & immutable / transformation (lazy evaluation)
  - Programming with RDD / DataFrame
    - Lambda function / transformations on RDDs / Actions on RDDs
  - Spark APIs - RDD / DataFrame / Dataset
  - Running SQL

Course Project

- Group size: 3 - 4 people (Email me your group information)
- Weight: 30%
- Deadline: Week 12
- Presentation: ~ 10 min (Date & Time: TBC)

Outline

- Semi-Structured Data
- Object Orientation
- Textual Data
- Spatial Data

Object Orientation

Object Orientation

- Relational database systems supports a small, fixed collection of data types (e.g., integers, dates, strings)
- Features offered by relational DBMS (RDBMS), for example, concurrency control and recovery, and query capabilities, become attractive and necessary for applications that need to handle more complex data
- Many database applications are written in object-oriented programming languages (e.g., Java, Python, C++) whose type system does not match that supported by RDBMS
  - Data need to be translated between the two models whenever they are fetched or stored.
  - Going through an intermediate language such as SQL is not always desirable
- The **Object-relational data model** extends the *relational data model* by providing a richer type system
  - With complex data types and object orientation

Object Orientation

- Approaches for integrating object-orientation with databases
  - Build an **object-relational database system**, adding object-oriented features to a relational database
  - Automatically convert data between programming language model and relational model for storage and retrieval; data conversion specified by **object-relational mapping**
  - Build an **object-oriented database system** that natively supports object-oriented data (type system) and direct access from programming language
    - Also called object database system, in which information is represented in the form of objects as used in object-oriented programming

Object Orientation

- Approaches for integrating object-orientation with databases
  - Build an **object-relational database system**, adding object-oriented features to a relational database
  - Automatically convert data between programming language model and relational model for storage and retrieval; data conversion specified by **object-relational mapping**
  - Build an **object-oriented database system** that natively supports object-oriented data (type system) and direct access from programming language
    - Also called object database system, in which information is represented in the form of objects as used in object-oriented programming
    - Declarative querying is very important for efficiently accessing data
    - Direct access to objects via pointers was found to result in increased risk of database corruption due to pointer errors

Object-Relational Database Systems

- How *object-oriented features* can be added to relational database systems?

- Supporting new data types!

- User-defined types
  - Object extensions to SQL allow creation of structured user-defined types, references to such types, and tables containing tuples of such types, e.g.,

```
create type Person
    (ID varchar(20) primary key,
     name varchar(20),
     address varchar(20))  ref from(ID);  /* More on this later */
create table people of Person;
```

  - We can create a new person as follows:

```
insert into people (ID, name, address) values
        ('12345', 'Susan', '23 Coyote Run');
```

Object-Relational Database Systems

- Table types (- Microsoft SQL Server)
  - **create type** *interest* **as table** (
    topic **varchar**(20),
    degree_of_interest **int**);
  - **create table** *users* (
    ID **varchar**(20),
    name **varchar**(20),
    interests interest);
- Array, multiset data types also supported by many databases
  - Syntax varies by database
  - ProgreSQL - *integer[]* denotes an array of integers whose size is not prespecified
  - Oracle - ***varray****(10)* ***of integer*** to specify an array of 10 integers

Type Inheritance

- Type inheritance
  - **create type** *Student* **under** *Person* (*degree* **varchar**(20)) ;
    create type *Teacher* **under** *Person* (*salary* **integer**);
- Both *Student* and *Teacher* inherit the attributes of *Person*—namely, ID, name, and address.
- *Student* and *Teacher* are said to be **subtypes** of *Person*, and
- *Person* is a **supertype** of *Student*, as well as of *Teacher*.

- User-defined types

```
create type Person
    (ID varchar(20) primary key,
     name varchar(20),
     address varchar(20))  ref from(ID);  /* More
on this later */

create table people of Person;
```

Table Inheritance

- Table inheritance syntax in PostgreSQL
  - **create table** *students*
    (*degree* **varchar**(20))
    inherits *people*;
    create table *teachers*
    (*salary* **integer**)
    inherits *people*;
  - Every attribute present in the table *people* is also present in the subtables *students* and *teachers*.

- User-defined types

create type *Person*
(*ID* **varchar**(20) **primary key**,
name **varchar**(20),
address **varchar**(20)) **ref from**(*ID*); /* More on this later */

create table *people* **of** *Person*;

Table Inheritance

- Table inheritance syntax in PostgreSQL
  - **create table** *students*
    (*degree* **varchar**(20))
    inherits *people*;
    create table *teachers*
    (*salary* **integer**)
    inherits *people*;
  - Every attribute present in the table *people* is also present in the subtables *students* and *teachers*.
- SQL:1999 - Oracle
  - Requires table types to be specified first
  - **create table** *students* **of** *Student*
    under *people*;
    create table *teachers* **of** *Teacher*
    under *people*;
- User-defined types

create type *Person*
(*ID* **varchar**(20) **primary key**,
name **varchar**(20),
address **varchar**(20)) **ref from**(*ID*); /* More on this later */

create table *people* **of** *Person*;

create type *Student* **under** *Person*
(*degree* **varchar**(20)) ;
create type *Teacher* **under** *Person*
(*salary* **integer**);

Table Inheritance

- Table inheritance syntax in PostgreSQL
  - **create table** *students*
    (*degree* **varchar**(20))
    inherits *people*;
    create table *teachers*
    (*salary* **integer**)
    inherits *people*;
  - Every attribute present in the table *people* is also present in the subtables *students* and *teachers*.
- SQL:1999 - Oracle
  - Requires table types to be specified first
  - **create table** *students* **of** *Student*
    under *people*;
    create table *teachers* **of** *Teacher*
    under *people*;

- User-defined types

create type *Person*
(*ID* **varchar**(20) **primary key**,
name **varchar**(20),
address **varchar**(20)) **ref from**(*ID*); /* More on this later */

create table *people* **of** *Person*;

create type *Student* **under** *Person*
(*degree* **varchar**(20)) ;
create type *Teacher* **under** *Person*
(*salary* **integer**);

insert into *student* **values**
('00128', 'Zhang', '235 Coyote Run', 'Ph.D.');

Reference Types

- Some SQL implementations such as Oracle support reference types.

- Creating reference types

  - **create type** *Person*
    (*ID* **varchar**(20) **primary key**,
    name **varchar**(20),
    address **varchar**(20))
    ref from(*ID*);
    create table *people* **of** *Person*;

    A reference-type declaration

  - By default, SQL assigns system-defined identifiers for tuples, but an existing primary key value can be used to reference a tuple by including the **ref from** clause in the type definition as shown above.

Reference Types

- Some SQL implementations such as Oracle support reference types.

- Creating reference types
  - **create type** *Person*
    (*ID* **varchar**(20) **primary key**,
    name **varchar**(20),
    address **varchar**(20))
    ref from(*ID*);
    create table *people* **of** *Person*;

  A reference-type declaration

  - **create type** *Department* (
    dept_name **varchar(20)**,
    head **ref**(*Person*) **scope** *people*);
    create table *departments* **of** *Department*

The **scope** clause restricts the scope of references to a single table.

head that is a reference to the type Person.

Reference Types

- Some SQL implementations such as Oracle support reference types.

- Creating reference types
  - **create type** *Person*
    (*ID* **varchar**(20) **primary key**,
    name **varchar**(20),
    address **varchar**(20))
    ref from(*ID*);
    create table *people* **of** *Person*;

    A reference-type declaration

  - **create type** *Department* (
    dept_name **varchar(20)**,
    head **ref**(*Person*) **scope** *people*);
    create table *departments* **of** *Department*

    The **scope** clause restricts the scope of references to a single table.

    head that is a reference to the type Person.

  - **insert into** *departments* **values** ('CS', '12345')

    insert into people (ID, name, address) **values**
    ('12345', 'Susan', '23 Coyote Run');

Reference Types

- Some SQL implementations such as Oracle support reference types.

- Creating reference types
  - **create type** *Person*
    (*ID* **varchar**(20) **primary key**,
    name **varchar**(20),
    address **varchar**(20))
    ~~**ref from**(*ID*);~~
    create table *people* **of** *Person*;

  - System generated references can be retrieved using **ref**(*r*)

    (*r* : table alias)
    - **select ref**(*p*)
      from *people* **as** *p*
      where *ID* = '12345';

Reference Types

- Some SQL implementations such as Oracle support reference types.

- Creating reference types
  - **create type** *Person*
    (*ID* **varchar**(20) **primary key**,
    name **varchar**(20),
    address **varchar**(20))
    ~~**ref from**(*ID*);~~
    create table *people* **of** *Person*;

  - System generated references can be retrieved using **ref**(*r*)
    (*r* : table alias)
    - **select ref**(*p*)
      from *people* **as** *p*
      where *ID* = '12345';

- **insert into** *departments* **values** ('CS', '12345')

- **insert into** *departments* **values** ('CS', null)
- **update** *departments*
  set *head* = (**select ref**(*p*)
  from *people* **as** *p*
  where *ID* = '12345')
  where *dept_name* = 'CS';

Reference Types

- Some SQL implementations such as Oracle support reference types.

- Creating reference types
  - **create type** *Person*
    (*ID* **varchar**(20) **primary key**,
    name **varchar**(20),
    address **varchar**(20))
    ~~**ref from**(*ID*);~~
    create table *people* **of** *Person*;

  - System generated references can be retrieved using **ref**(*r*)
    (*r* : table alias)
    - **select ref**(*p*)
      from *people* **as** *p*
      where *ID* = '12345';

Most database systems do not allow subqueries in an insert into departments values statement,

- **insert into** *departments* **values** ('CS', '12345')
- **insert into** *departments* **values** ('CS', null)
- **update** *departments*
  set *head* = (**select ref**(*p*)
  from *people* **as** *p*
  where *ID* = '12345')
  where *dept_name* = 'CS';

Reference Types

- References are dereferenced in SQL:1999 by the **–>** symbol.
  - **create type Department (**
    dept_name varchar(20),
    head ref(Person) scope people);
    create table departments of Department

- Using references in path expressions
  - **select** *head->name*, *head->address*
    from *departments*;

Reference Types

- References are dereferenced in SQL:1999 by the **–>** symbol.
  - **create type Department (**
    dept_name varchar(20),
    head ref(Person) scope people);
    create table departments of Department

- Using references in path expressions
  - **select** *head->name*, *head->address*
    from *departments*;

An expression such as "head->name" is called a path expression.

the name attribute of the tuple from the people table

  - To find the name and address of the head of a department, we would require an explicit join of the relations departments and people. The use of references *simplifies* the query considerably.

Reference Types

- References are dereferenced in SQL:1999 by the **–>** symbol.
  - **create type Department (**
    dept_name varchar(20),
    head ref(Person) scope people);
    create table departments of Department

- Using references in path expressions
  - **select** *head->name*, *head->address*
    from *departments*;

An expression such as "head –>name" is called a path expression.

the name attribute of the tuple from the people table

  - To find the name and address of the head of a department, we would require an explicit join of the relations departments and people. The use of references simplifies the query considerably.

- Use operation **deref** to return the tuple pointed to by a reference
  - **select deref***(head).name*

    from *departments*;

Array and Multiset Types in SQL

- Example of array and multiset declaration:

```
create type Publisher as
    (name            varchar(20),
     branch          varchar(20));
create type Book as
    (title           varchar(20),
     author_array    varchar(20) array [10],
     pub_date        date,
     publisher       Publisher,
     keyword-set     varchar(20) multiset);
create table books of Book;
```

title	*author_array*	*publisher*	*keyword_set*
		(*name*, *branch*)	
Compilers	[Smith, Jones]	(McGraw-Hill, NewYork)	{parsing, analysis}
Networks	[Jones, Frick]	(Oxford, London)	{Internet, Web}

Object-Relational Mapping

- Object-relational mapping (ORM) systems allow
  - Specification of mapping between programming language objects and database tuples
  - Automatic creation of database tuples upon creation of objects
  - Automatic update/delete of database tuples when objects are update/deleted

  - Provide query languages that allow programmers to write queries directly on the object model
    - Such such queries are translated into SQL queries on the underlying relational database
    - Result objects are created from the tuples in the SQL query results

  - Pros
    - ORMs hide minor SQL differences between databases from the higher levels
  - Cons
    - May suffer from significant performance inefficiencies for bulk database updates, as well as for complex queries that are not written in SQL

Textual Data

Textual Data

- Textual data are unstructured

- **Information retrieval**: querying of unstructured data
  - Simple model of **keyword queries**: given query keywords, retrieve documents containing all the keywords
  - More advanced models rank relevance of documents
  - Today, keyword queries return many types of information as answers
    - E.g., a query “cricket” typically returns information about ongoing cricket matches
    - E.g., a query “New York”, a search engine may show a map of New York, and images of New York, in addition to web pages related to New York.

- **Relevance ranking**
  - Essential since there are usually many documents matching keywords

Information Retrieval Systems vs Database Systems

- **Information retrieval (IR)** systems use a simpler data model than database systems
  - Information organized as a collection of unstructured documents
- IR systems deal with some querying issues not generally addressed by database systems
  - Approximate searching by keywords
  - Ranking of retrieved answers by estimated degree of relevance
  - Web search engines are the most familiar example of IR systems

- Database systems deal with structured information organized with relatively complex data models
  - E.g., the relational model or object-oriented data models
- Database systems deal with transactional updates (including concurrency control and recovery)
  - Less important in IR systems

Ranking using TF-IDF

- The word "**term**" refers to a keyword occurring in a document/query
- To address: Given a particular term *t*, how relevant is a particular document *d* to the term?
- One approach - use the number of occurrences of term *t* in document *d* as a measure of its relevance
  - The number of occurrences depends on the length of the document
  - A document containing 10 occurrences of a term may not be 10 times as relevant as a document containing one occurrence

Ranking using TF-IDF

- **Term Frequency:** $TF(d, t)$, the relevance of a term t to a document d
  - One definition: $TF(d, t) = log(1 + n(d,t)/n(d))$
    where
    - $n(d,t)$ = number of occurrences of term t in document d
    - $n(d)$ = number of terms in document d
    - This metric takes the length of the document into account.
    - The relevance grows with more occurrences of a term in the document, but not directly proportional to the number of occurrences.

- Given a keyword query Q containing multiple keywords {*t1, t2*}, how to combining the relevance measures of the document *d* for each keyword (i.e., TF(*d, t1*), TF(*d, t2*)}?

Ranking using TF-IDF

- **Term Frequency:** $TF(d, t)$, the relevance of a term t to a document d
  - One definition: $TF(d, t) = log(1 + n(d,t)/n(d))$
    where
    - $n(d,t)$ = number of occurrences of term t in document d
    - $n(d)$ = number of terms in document d
- **Inverse document frequency**: IDF(t)
  - One definition: $IDF(t) = 1/n(t)$
    - $n(t)$ = number of documents that contain the term t

- **Relevance** of a document d to a set of terms Q
  - *One definition:* $r(d, Q) = \sum_{t \in Q} TF(d, t) * IDF(t)$
  - Other definitions
    - Take **proximity** of words into account
    - **Stop words** (e.g., “and”, “or”, “a”, “the”, and etc.) are often ignored

Ranking Using Hyperlinks

- **Hyperlinks** between documents can be used to decide on the overall importance of a document, *independent of the keyword query*
  - Documents linked from many other documents are considered more important

- Google introduced **PageRank**, a measure of popularity/importance based on hyperlinks to pages
  - Pages hyperlinked from many pages should have higher PageRank
  - Pages hyperlinked from pages with higher PageRank should have higher PageRank
  - Formalized by **random walk** model

- Let *T*[*i*, *j*] be the probability that a random walker who is on page *i* will click on the link to page *j*
  - Assuming all links are equal, *T*[*i*, *j*] = 1/*Ni* (Ni = the number of links out of page i)
- Then PageRank[*j*] for each page *j* can be defined as

$$P[j] = \delta/N + (1 - \delta) * \sum_{i=1}^{N} (T[i,j] * P[i])$$

  - Where *N* = total number of pages, and δ (0< δ <1) a constant usually set to 0.15

Ranking Using Hyperlinks

- Definition of PageRank is circular, but can be solved as a set of linear equations
  - Simple iterative technique works well
  - Initialize all P[i] = 1/N
  - In each iteration use equation $P[j] = \delta/N + (1 - \delta) * \sum_{i=1}^{N} (T[i, j] * P[i])$ to update P
  - Stop iteration when changes are small, or some limit (say 30 iterations) is reached.

- Other measures of relevance are also important. For example:
  - Keywords in anchor text
  - Number of times uses click on a link if it is returned as an answer

Measuring Retrieval Effectiveness

- Two metrics
  - **Precision**: what percentage of retrieved documents are actually relevant

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

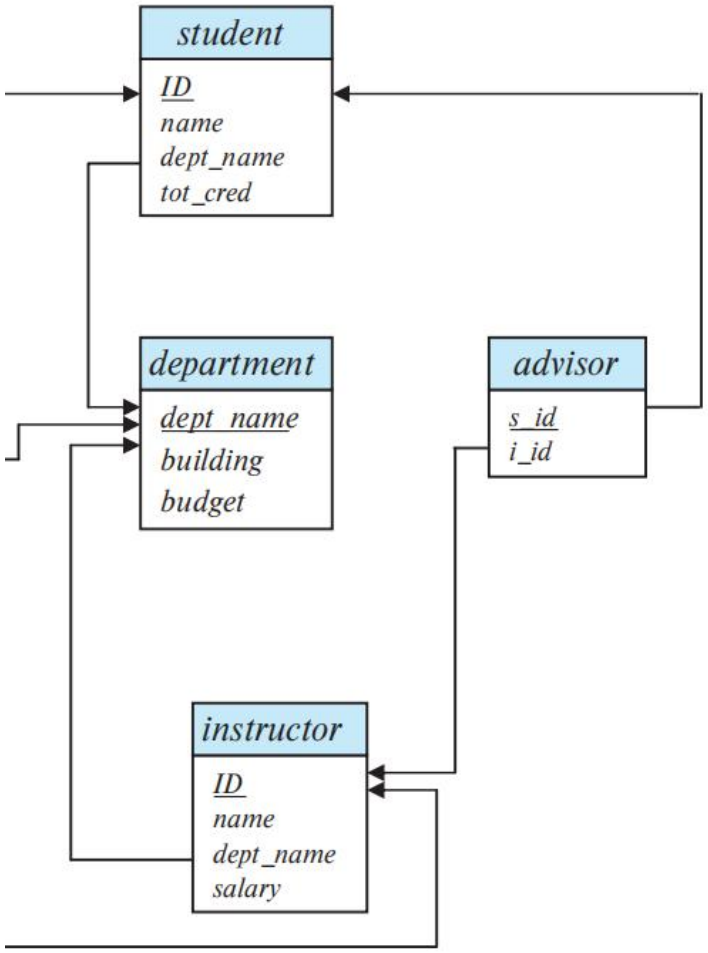
  - **Recall**: what percentage of relevant documents were returned

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

  - At some number of answers, e.g. precision@k, recall@k
    - precision@k = (# relevant documents in top-k results) / k
    - recall@k = (# relevant documents in top-k results) / (total # relevant documents)

Keyword Search in Relational Databases

- Keyword querying on structured data
  - Useful if users don't know schema (*complex* or *missing*) and SQL
  - Keywords match tuples
  - Keyword search returns closely connected tuples that contain all keywords
    - E.g. on our university database given query "Zhang Katz",
    - "Zhang" matches a student
    - "Katz" matches an instructor
    - *advisor* relationship links them



Keyword Search in Relational Databases

VLDB 2002

DISCOVER: Keyword Search in Relational Databases

Vagelis Hristidis
University of California, San Diego
vagelis@cs.ucsd.edu

Yannis Papakonstantinou
University of California, San Diego
yannis@cs.ucsd.edu

ORDERS

	ORDERKEY	CUSTKEY	ORDERSTATUS	TOTALPRICE	ORDERDATE	ORDERPRIORITY	CLERK	...
o_1	1000105	12312	complete	$5,000	5/2/2001	High	John Smith	
o_2	1000111	12312	in process	$3,000	5/1/2001	High	Mike Miller	
o_3	1000125	10001	in process	$7,000	5/1/2001	Low	Mike Miller	
o_4	1000110	10002	complete	$8,000	4/25/2001	Low	Keith Brown	

CUSTOMER

	CUSTKEY	NAME	ADDRESS	NATIONKEY	PHONE	...
c_1	12312	Brad Lou	3811 State Drive, Los Angeles	01	454-1234567	
c_2	10001	George Walters	4365 5th Ave, New York	01	561-2345678	
c_3	10013	John Roberts	3234 Broadway St, San Francisco	01	643-5473921	

NATION

	NATIONKEY	NAME	REGIONKEY	COMMENT
n_1	01	USA	N.America	null

LINEITEM

	ORDERKEY	PARTKEY	SUPPKEY	LINENUMBER	...
l_1	1000105	1122	111222	2	
l_2	1000110	1122	111222	4	
l_3	1000110	2233	222333	3	
l_4	1000111	2233	222333	2	

PARTSUPP

	PARTKEY	SUPPKEY	AVAILQTY	...
p_1	1122	111222	1000	
p_2	2233	222333	400	

Sample TPC-H database instance (www.tpc.org)

- Keyword query = { “Smith”, “Miller”}

Keyword Search in Relational Databases

VLDB 2002

DISCOVER: Keyword Search in Relational Databases

Vagelis Hristidis
University of California, San Diego
vagelis@cs.ucsd.edu

Yannis Papakonstantinou
University of California, San Diego
yannis@cs.ucsd.edu

ORDERS

	ORDERKEY	CUSTKEY	ORDERSTATUS	TOTALPRICE	ORDERDATE	ORDERPRIORITY	CLERK	...
o_1	1000105	12312	complete	$5,000	5/2/2001	High	John Smith	
o_2	1000111	12312	in process	$3,000	5/1/2001	High	Mike Miller	
o_3	1000125	10001	in process	$7,000	5/1/2001	Low	Mike Miller	
o_4	1000110	10002	complete	$8,000	4/25/2001	Low	Keith Brown	

CUSTOMER

	CUSTKEY	NAME	ADDRESS	NATIONKEY	PHONE	...
c_1	12312	Brad Lou	3811 State Drive, Los Angeles	01	454-1234567	
c_2	10001	George Walters	4365 5th Ave, New York	01	561-2345678	
c_3	10013	John Roberts	3234 Broadway St, San Francisco	01	643-5473921	

NATION

	NATIONKEY	NAME	REGIONKEY	COMMENT
n_1	01	USA	N.America	null

LINEITEM

	ORDERKEY	PARTKEY	SUPPKEY	LINENUMBER	...
l_1	1000105	1122	111222	2	
l_2	1000110	1122	111222	4	
l_3	1000110	2233	222333	3	
l_4	1000111	2233	222333	2	

PARTSUPP

	PARTKEY	SUPPKEY	AVAILQTY	...
p_1	1122	111222	1000	
p_2	2233	222333	400	

Sample TPC-H database instance (www.tpc.org)

- Keyword query = { “Smith”, “Miller”}
- Two *minimal* joining sequences
  - o1 ⋈ c1 ⋈ o2
  - o1 ⋈ c1 ⋈ n1 ⋈ c2 ⋈ o3
  - **Minimal** = no tuple can be excluded and still have a sequence that contains the keywords

Keyword Search in Relational Databases

VLDB 2002

DISCOVER: Keyword Search in Relational Databases

Vagelis Hristidis
University of California, San Diego
vagelis@cs.ucsd.edu

Yannis Papakonstantinou
University of California, San Diego
yannis@cs.ucsd.edu

ORDERS

	ORDERKEY	CUSTKEY	ORDERSTATUS	TOTALPRICE	ORDERDATE	ORDERPRIORITY	CLERK	...
o_1	1000105	12312	complete	$5,000	5/2/2001	High	John Smith	
o_2	1000111	12312	in process	$3,000	5/1/2001	High	Mike Miller	
o_3	1000125	10001	in process	$7,000	5/1/2001	Low	Mike Miller	
o_4	1000110	10002	complete	$8,000	4/25/2001	Low	Keith Brown	

CUSTOMER

	CUSTKEY	NAME	ADDRESS	NATIONKEY	PHONE	...
c_1	12312	Brad Lou	3811 State Drive, Los Angeles	01	454-1234567	
c_2	10001	George Walters	4365 5th Ave, New York	01	561-2345678	
c_3	10013	John Roberts	3234 Broadway St, San Francisco	01	643-5473921	

NATION

	NATIONKEY	NAME	REGIONKEY	COMMENT
n_1	01	USA	N.America	null

LINEITEM

	ORDERKEY	PARTKEY	SUPPKEY	LINENUMBER	...
l_1	1000105	1122	111222	2	
l_2	1000110	1122	111222	4	
l_3	1000110	2233	222333	3	
l_4	1000111	2233	222333	2	

PARTSUPP

	PARTKEY	SUPPKEY	AVAILQTY	...
p_1	1122	111222	1000	
p_2	2233	222333	400	

Sample TPC-H database instance (www.tpc.org)

- Keyword query = { “Smith”, “Miller”}
- Two *minimal* joining sequences
  - o1 ⋈ c1 ⋈ o2
  - o1 ⋈ c1 ⋈ n1 ⋈ c2 ⋈ o3
  - **Minimal** = no tuple can be excluded and still have a sequence that contains the keywords
- Semantics of the two joining sequences
  - S1 shows that both “Smith” and “Miller” are clerks that have served customer *Brad Lou*
  - S2 shows that the clerks have served customers *Brad Lou* and *George Walters* respectively, who both come from the USA.

Keyword Search in Relational Databases

VLDB 2002

DISCOVER: Keyword Search in Relational Databases

Vagelis Hristidis
University of California, San Diego
vagelis@cs.ucsd.edu

Yannis Papakonstantinou
University of California, San Diego
yannis@cs.ucsd.edu

ORDERS

	ORDERKEY	CUSTKEY	ORDERSTATUS	TOTALPRICE	ORDERDATE	ORDERPRIORITY	CLERK	...
o_1	1000105	12312	complete	$5,000	5/2/2001	High	John Smith	
o_2	1000111	12312	in process	$3,000	5/1/2001	High	Mike Miller	
o_3	1000125	10001	in process	$7,000	5/1/2001	Low	Mike Miller	
o_4	1000110	10002	complete	$8,000	4/25/2001	Low	Keith Brown	

CUSTOMER

	CUSTKEY	NAME	ADDRESS	NATIONKEY	PHONE	...
c_1	12312	Brad Lou	3811 State Drive, Los Angeles	01	454-1234567	
c_2	10001	George Walters	4365 5th Ave, New York	01	561-2345678	
c_3	10013	John Roberts	3234 Broadway St, San Francisco	01	643-5473921	

NATION

	NATIONKEY	NAME	REGIONKEY	COMMENT
n_1	01	USA	N.America	null

LINEITEM

	ORDERKEY	PARTKEY	SUPPKEY	LINENUMBER	...
l_1	1000105	1122	111222	2	
l_2	1000110	1122	111222	4	
l_3	1000110	2233	222333	3	
l_4	1000111	2233	222333	2	

PARTSUPP

	PARTKEY	SUPPKEY	AVAILQTY	...
p_1	1122	111222	1000	
p_2	2233	222333	400	

Sample TPC-H database instance (www.tpc.org)

- Keyword query = { “Smith”, “Miller”}
- Two *minimal* joining sequences
  - o1 ⋈ c1 ⋈ o2
  - o1 ⋈ c1 ⋈ n1 ⋈ c2 ⋈ o3
  - **Minimal** = no tuple can be excluded and still have a sequence that contains the keywords
- Semantics of the two joining sequences
  - S1 shows that both “Smith” and “Miller” are clerks that have served customer *Brad Lou*
  - S2 shows that the clerks have served customers *Brad Lou* and *George Walters* respectively, who both come from the USA.

Spatial Data

Spatial Data

- *Spatial data support* in database systems is important for efficiently storing, indexing and querying of data on the basis of spatial locations.

- Two types of spatial data are particularly important
  - **Geographic data** -- road maps, land-usage maps, topographic elevation maps, political maps showing boundaries, land-ownership maps, and so on.
    - **Geographic information systems** are special-purpose databases tailored for storing geographic data.
    - Round-earth coordinate system may be used
      - (Latitude, longitude, elevation)

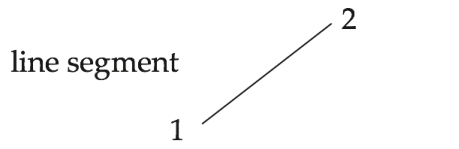
  - **Geometric data:** design information about how objects are constructed. For example, designs of buildings, aircraft, layouts of integrated-circuits.
    - Based on 2 or 3 dimensional Euclidean space with (X, Y, Z) coordinates

Representation of Geometric Information

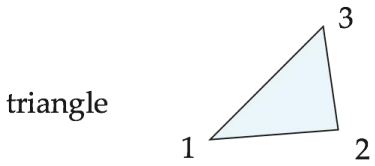
Various geometric constructs can be represented in a database in a normalized fashion (see next slide)

- A **line segment** can be represented by the coordinates of its endpoints.
- A **polyline** or **linestring** consists of a connected sequence of line segments and can be represented by a list containing the coordinates of the endpoints of the segments, in sequence.
  - Approximate a curve by partitioning it into a sequence of segments
    - Useful for two-dimensional features such as roads in a map.
    - Some systems also support *circular arcs* as primitives, allowing curves to be represented as sequences of arc
- **Polygons** is represented by a list of vertices in order.
  - The list of vertices specifies the boundary of a polygonal region.
  - Can also be represented as a set of triangles (**triangulation**)

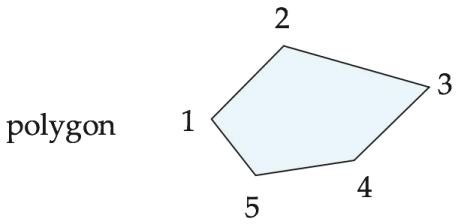
Representation of Geometric Constructs



{(x1,y1), (x2,y2)}

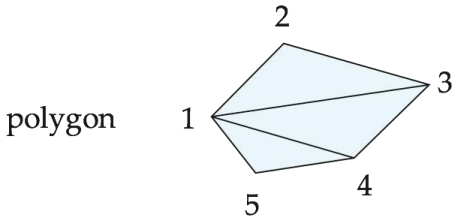


{(x1,y1), (x2,y2), (x3,y3)}



{(x1,y1), (x2,y2), (x3,y3), (x4,y4), (x5,y5)}

Triangulation



{(x1,y1), (x2,y2), (x3,y3), ID1}
{(x1,y1), (x3,y3), (x4,y4), ID1}
{(x1,y1), (x4,y4), (x5,y5), ID1}

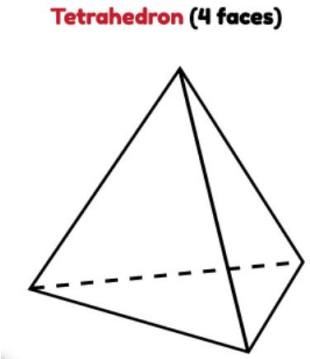
Identifier of the polygon

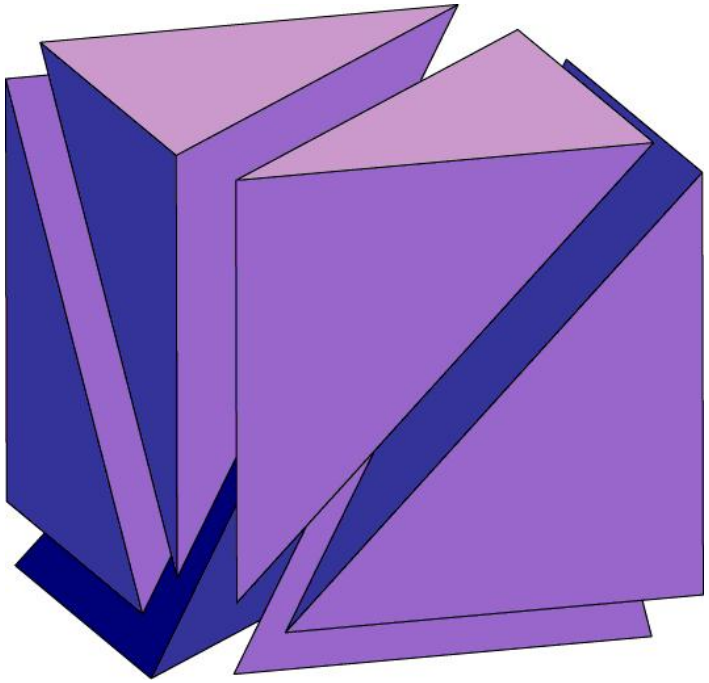
object **representation**

Representation of Geometric Information (Cont.)

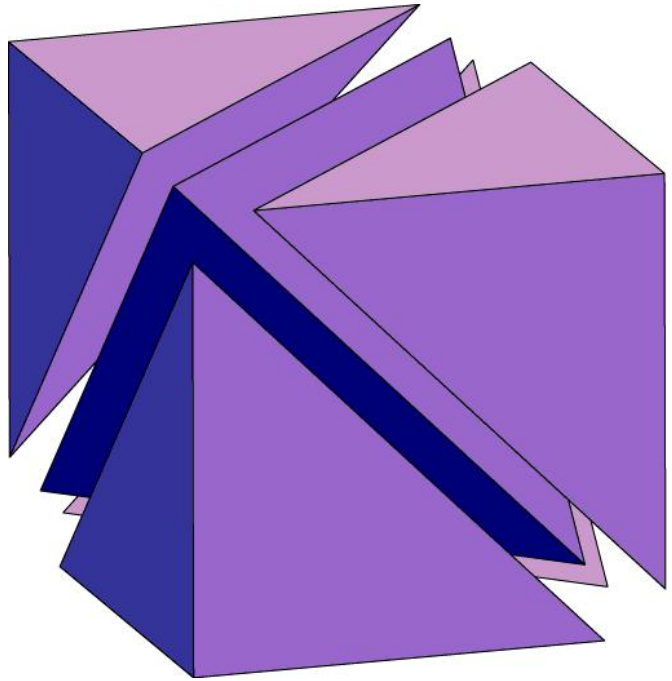
- Representation of points and line segment in **3-D space** similar to 2-D, except that points have an extra *z* component
- Represent arbitrary *polyhedra* by dividing them into *tetrahedrons*, like triangulating polygons.
  - Alternative: List their faces, each of which is a polygon, along with an indication of which side of the face is inside the polyhedron.

Any n-vertex convex polyhedron can be divided into O(n) tetrahedra





Triangulating the cube into 6 tetrahedra



Triangulating the cube into 5 tetrahedra

https://www.ics.uci.edu/~eppstein/projects/tetra/#:~:text=Any%20n%2Dvertex%20convex%20polyhedron,always%20the%20best%20possible%20triangulation.

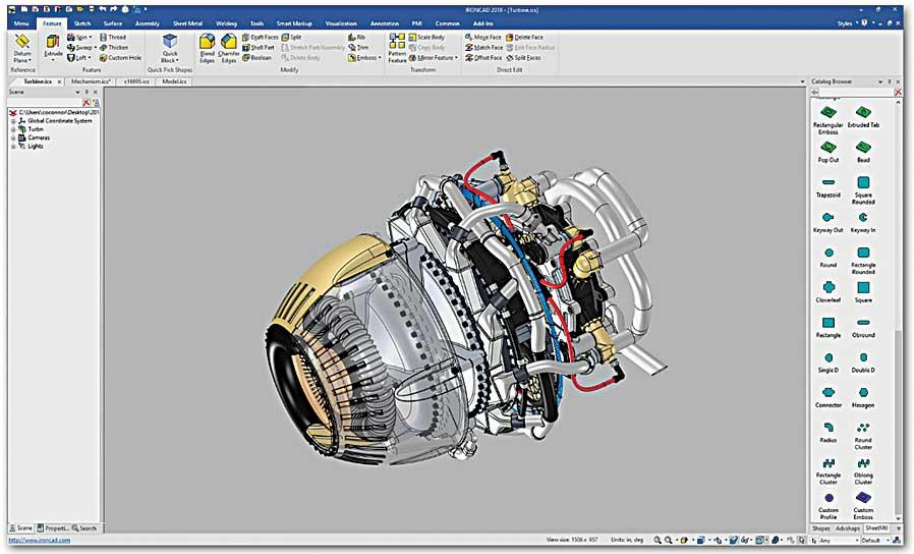
Representation of Geometric Information (Cont.)

- *Geographic* and *geometric* data types are supported by many database systems, such as Oracle Spatial and Graph, the PostGIS extension of PostgreSQL, SQL Server, and the IBM DB2 Spatial Extender.

- E.g. SQL Server and PostGIS
  - Types: point, linestring, curve, polygons
  - Collections: multipoint, multilinestring, multicurve, multipolygon
  - Textual representations:
    - LINESTRING(1 1, 2 3, 4 4) - defines a line that connects points (1, 1), (2, 3) and (4, 4)
    - POLYGON((1 1, 2 3, 4 4, 1 1)) - defines a triangle defined by these points
  - Type conversions: *ST GeometryFromText*() and *ST GeographyFromText*()
    - Convert the textual representations to geometry and geography objects respectively
  - *Operations: ST Union*(), *ST Intersection*()
    - Compute the union and intersection of geomet ric objects such as linestrings and polygons

Design Databases

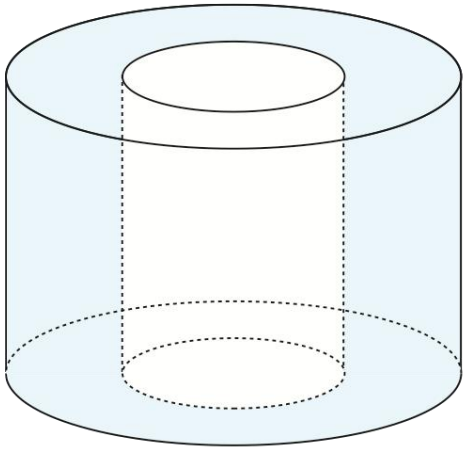
- *Computer-aided-design* (CAD) systems traditionally stored data in memory during editing or other processing and wrote the data back to a file at the end of a session of editing.
  - Read in an entire file even if only parts of it are required
  - Impossible to hold large design in memory
- Designers of object-oriented databases were motivated in large part by the database requirements of CAD systems.
- Object-oriented databases represent design components as objects (generally geometric objects); the connections between the objects indicate how the design is structured.



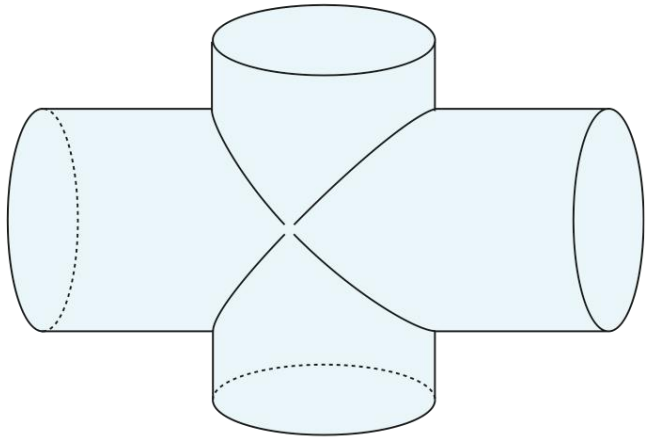
https://www.assemblymag.com/ext/resources/Issues/2020/January/nn/asb0120nn2.jpg?1579617111

Representation of Geometric Constructs

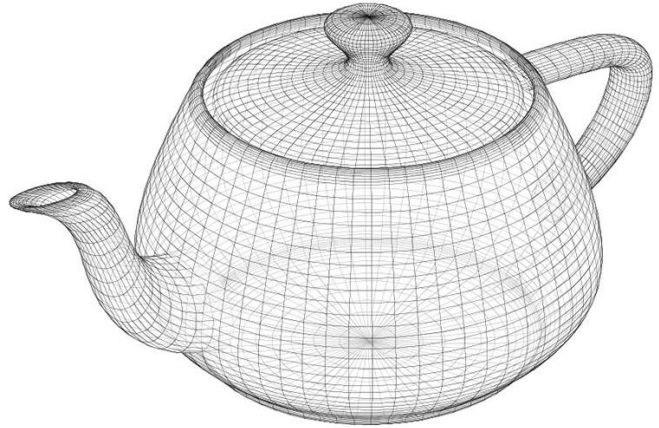
- Object-oriented databases represent design components as objects (generally geometric objects); the connections between the objects indicate how the design is structured.
  - Simple two-dimensional objects: points, lines, triangles, rectangles, polygons.
  - Complex two-dimensional objects: formed from simple objects via union, intersection, and difference operations.
  - Complex three-dimensional objects: formed from simpler objects such as spheres, cylinders, and cuboids, by union, intersection, and difference operations (e.g., (a) and (b)).
  - ***Wireframe models*** represent three-dimensional surfaces as a set of simpler objects (c).



(a) Difference of cylinders



(b) Union of cylinders



(c) Wireframe model of a teapot

https://techterms.com/img/sm/wireframe_1280-2.jpg

Design Databases

- Design databases also store non-spatial information about objects
  - E.g., construction material, color, etc.
  - Can model by standard data-modeling techniques

- Spatial-integrity constraints are important.
  - E.g., two pipes should not be in the same location, wires should not be too close to each other, etc.
  - Database support for spatial-integrity constraints helps to avoid design errors, keeping the design consistent

Geographic Data

- Geographic data can be categorized into two types

- **Raster data** consist of bit maps or pixel maps, in two or more dimensions.
  - Example 2-D raster image: satellite image of cloud cover, where each pixel stores the cloud visibility in a particular area.
    - In addition to the image, the data include the location of the image (e.g., latitude and longitude of its corners), and the resolution
  - Additional dimensions might include the temperature at different *altitudes* at different regions, or measurements taken at different *points in time*.

- Design databases generally do not store raster data.

Geographic Data (Cont.)

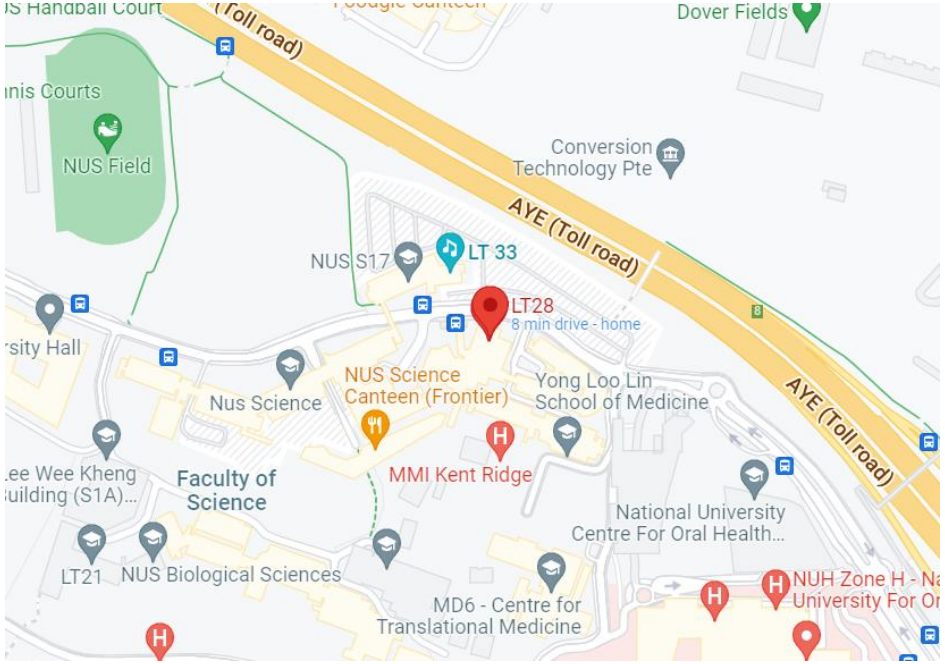
- Geographic data can be categorized into two types

- **Vector data** are constructed from basic geometric objects: points, line segments, triangles, and other polygons in two dimensions, and cylinders, spheres, cuboids, and other polyhedrons in three dimensions.
  - In the context of geographic data, points are usually represented by latitude and longitude, and where the height is relevant, additionally by elevation.

- Vector format often used to represent map data.
  - Roads can be considered as two-dimensional and represented by lines and curves.
  - Some features, such as rivers, may be represented either as complex curves or as complex polygons, depending on whether their width is relevant.
  - Features such as regions and lakes can be depicted as polygons.

Applications of Geographic Data

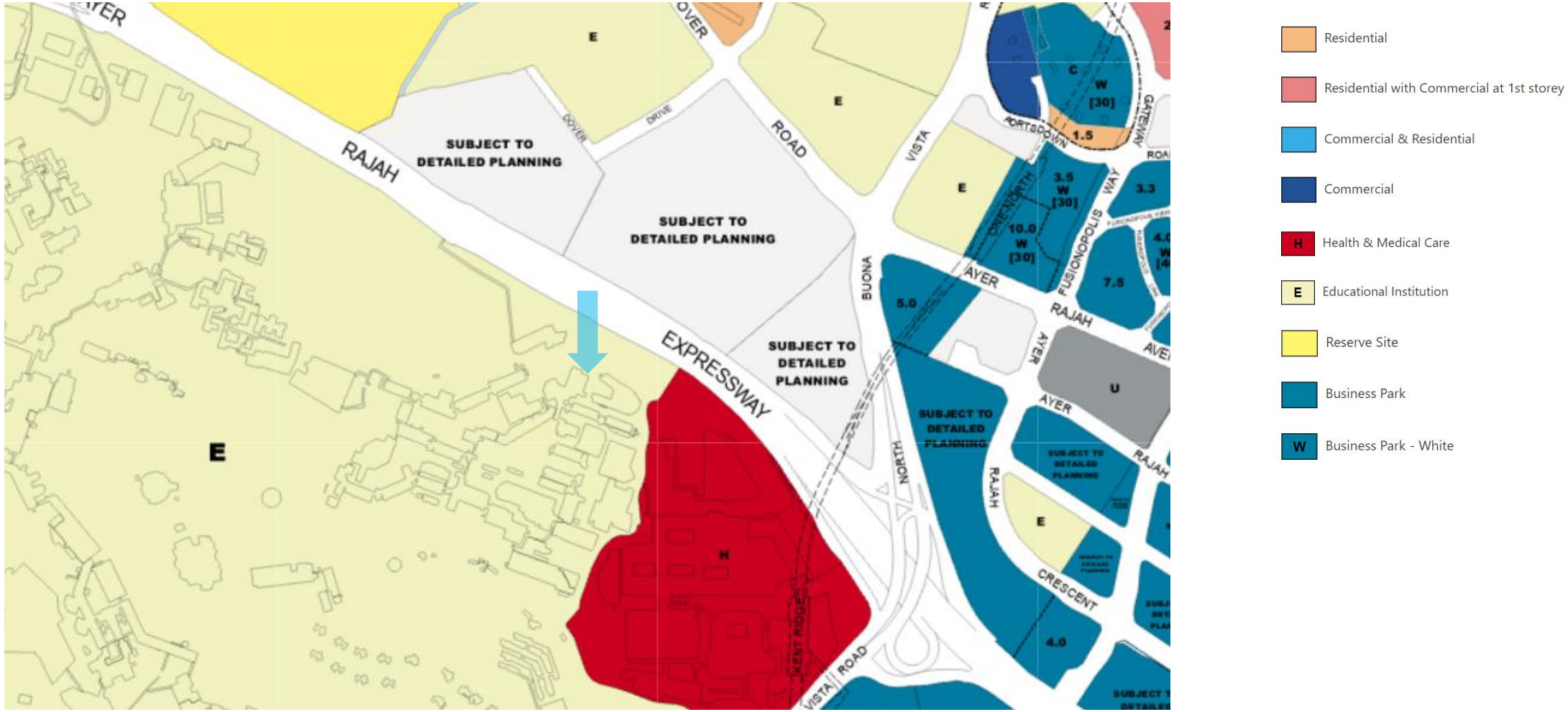
- Geographic databases have a variety of uses, including online map and navigation services, which are ubiquitous today.



Applications of Geographic Data

Applications of Geographic Data

- Master Plan 2019 - land use plan that will guide Singapore's developments over the next 10 to 15 years



Spatial Queries

There are a number of types of queries that involve spatial locations (with a combination of spatial and nonspatial requirements)

- **Region queries** deal with spatial regions. e.g., ask for objects that lie partially or fully inside a specified region
  - E.g., PostGIS - *ST_Contains*(), *ST_Overlaps*(), …
- **Nearness queries** request objects that lie near a specified location, e.g., restaurants nearby
- **Nearest neighbor queries**, given a point or an object, find the nearest object that satisfies given conditions.
- **Spatial graph queries** request information based on spatial graphs (e.g., maps)
  - E.g., shortest path between two points via a road network
- **Spatial join** of two spatial relations with the location playing the role of join attribute.

Summary

- There are many application domains that need to store more complex data than simple tables with a fixed number of attributes.
- The SQL standard includes extensions of the SQL data-definition and query language to deal with new data types and with object orientation. These include support for collection-valued attributes, inheritance, and tuple references. Such extensions attempt to preserve the relational foundations—in particular, the declarative access to data—while extending the modeling power.
- The object-relational data model extends the relational data model by providing a richer type system, including collection types and object orientation.
- Object-relational database systems (i.e., database systems based on the object-relational model) provide a convenient migration path for users of relational databases who wish to use object-oriented features.

Summary (Cont.)

- Information-retrieval systems are used to store and query textual data such as documents. They use a simpler data model than do database systems but provide more powerful querying capabilities within the restricted model.
- Queries attempt to locate documents that are of interest by specifying, for example, sets of keywords. The query that a user has in mind usually cannot be stated precisely; hence, information-retrieval systems order answers on the basis of potential relevance.
- Relevance ranking makes use of several types of information, such as:
  - Term frequency: how important each term is to each document.
  - Inverse document frequency.
  - Popularity ranking.
- Spatial data management is important for many applications. Geometric and geographic data types are supported by many database systems, with subtypes including points, linestrings and polygons. Region queries, nearest neighbor queries, and spatial graph queries are among the commonly used types of spatial queries.

Homework 2

- MySQL + xml + Spark SQL
- Data: MAS Database + DBLP data (xml)
- Task - Answer queries over MAS + DBLP using Spark SQL
- Deadline: 2 weeks