Week 8 – Advanced Topics 2
Improving Database Performance

Dr Claire Ellul
c.ellul@ucl.ac.uk

Spatial Data Management

- Overview
  - Indexing
  - Normalisation and De-Normalisation
  - Query Parsing

Indexes

- Indexes
  - Provide accelerated access to data by reducing the set of objects to be looked for when processing a query
  - In simple terms, they improve query performance

Indexes

- Indexes
  - Take the following PERSONAL_DETAILS table

Name	Surname	Date of Birth
James	Jones	25th January 1987
James	Smith	13th March 1956
Robert	Jones	15th April 1945
Janet	Jones	15th April 1967
Richard	Jones	20th September 1972
Alex	Smith	19th November 1966
John	Ward	1st August 1979

Indexes

- Run the following query

```
SELECT *
FROM PERSONAL_DETAILS
WHERE SURNAME = 'SMITH'
```

Indexes

- Without an index
  - The system has to check each row to see if the surname is Smith
  - In a large table this can take a while!

Indexes

- The system has to run the following code

```
FOR EACH ROW IN THE TABLE
    IF ROW.SURNAME = 'SMITH' THEN
        ADD ROW TO RESULTS LIST
    END IF
NEXT ROW
```

- For the small table shown above this requires a total of 3 x 7 = 21 operations (3 operations – FOR, IF, ADD and 7 rows of data)

Indexes

- Without an index
  - This process must be repeated for each of the rows on the table – in the case of a table with 2 million
  - This results in 2 million x 3 = 6 million operations, which is not very efficient!

Indexes

- Adding an Index
  - This will speed up the process by creating a 'shortcut' to each unique surname in the table
  - Conceptually, the index will look something like this:

Surname	Rows
Jones	1,3,4,5
Ward	7
Smith	2,6

Indexes

- With an Index
  - The system now has to run the following code

```
FOR EACH ROW IN THE INDEX TABLE
    IF INDEX.SURNAME = 'SMITH' THEN
        FOR EACH ITEM IN THE LIST
            SELECT ROW FROM THE TABLE
            ADD THE DATA INTO THE RESULTS LIST
        NEXT ITEM IN THE LIST
    END IF
NEXT ROW IN THE INDEX TABLE
```

  - For the table above, that is a total of 12 operations!

Indexes

- Creating an Index in SQL
  - The SQL command to create an index is:

```
CREATE INDEX <INDEX_NAME>
ON <TABLENAME>(<FIELDNAME>);
```

  - For example:

```
CREATE INDEX SURNAME_IDX
ON PERSONAL_DETAILS(SURNAME);
```

Indexes

- Indexes
  - In reality, indexes are stored using a system called a B-Tree
  - This takes advantage of the structure of the hard disk of the computer to allow fast retrieval of the index data

Indexes

- When to use indexes
  - Non-spatial indexes are best used when some of the data in the column is the same
  - Indexes are not very efficient:
    - When each item in the column is unique
    - When the datasets are small (indexes will not make much difference in performance)

Indexes

- When to use Indexes
  - When deciding whether to use an index, you also need to look at how the data will be used
  - Will it be added to or updated very frequently?
  - Or will it be used for decision making – I.e. to answer queries?

Indexes

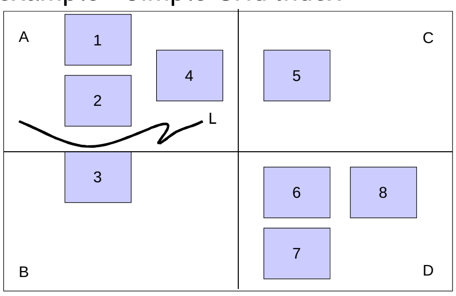
- When to use Indexes
  - Each time you insert, update or delete a record, the index has to be modified
  - This takes time, and the index may not be worth using if the data is not being used for decision support

Spatial Indexing

- Spatial indexes work on a similar principle to normal indexes
- Instead of looking for rows with the same surname, they look for rows with data that is in the same area
- The idea is that if you are interested in data for London, you will not be interested in data for Scotland, so the system should just get the London data you need without searching the Scotland data too

Indexing and Spatial Indexing

- An example – Simple Grid Index



Indexing and Spatial Indexing

- Grid Structure
  - This involves partitioning the space into regular rectangular grid squares called cells
  - A point is assigned to one of the grid cells if it is within the grid cell
  - Points, lines and polygons may be assigned to multiple grid cells if they overlap

UCL

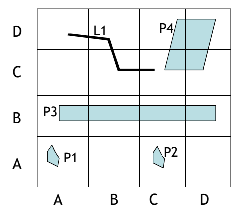
Indexing and Spatial Indexing

- Grid Structure
  - This is most efficient for managing point datasets
  - It works by storing links to items in one grid cell next to each other on the computer hard disk (in the sub-area of the disk called a page)
  - For convenience, we represent each page as a row in a table in the following diagram

UCL

Indexing and Spatial Indexing

- Grid Structure



Grid Cell	Contents
AA	P1
AB	
AC	P2
AD	
BA	P3
BB	P3
BC	P3
BD	P3

Grid Cell	Contents
CA	
CB	L1
CC	L1,P4
CD	P4
DA	L1
DB	L1
DC	P4
DD	P4

Note: Each ‘Grid Cell’ row in the above table actually corresponds to a page on the hard disk of the computer, rather than to a table in a database

UCL

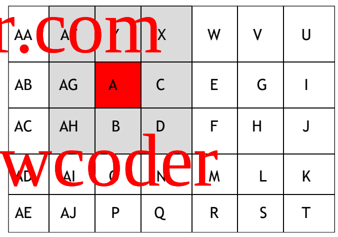
Indexing and Spatial Indexing

- Problems with the Grid Index (1)
  - Taking only ONE grid cell into account does not allow Polygon 3 to be included in the search for the polygon closest to line L
  - Therefore, expand the search to the grid cell containing L and all the surrounding cells
  - But this has performance implications!

UCL

Indexing and Spatial Indexing

- Expanded Grid Cell Search



UCL

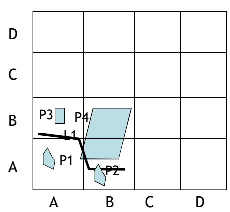
Indexing and Spatial Indexing

- Problems with the Grid Index (2)
  - What happens if all the data is in one corner of the given area?

UCL

Indexing and Spatial Indexing

- Grid Indexes



Grid Cell	Contents
AA	P1, L1,P4
AB	P2,L1,P4
AC	
AD	
BA	P3, L1
BB	P4
BC	
BD	

Grid Cell	Contents
CA	
CB	
CC	
CD	
DA	
DB	
DC	
DD	

-The grid has many empty cells, and a few cell that are densely packed

Indexing and Spatial Indexing

- Problems with the Grid Index (3) :
  - Index size can grow quite quickly, leading to an increase in search time and reduction in performance
    - Because lines and polygons overlap multiple grid squares, the index can grow to quite a large size - this is why it is most suited to point data
    - If a point falls on the intersection of four grids, it is assigned to all four, also increasing the size of the index
    - If the geometry is only distributed in a few cells, the other cells are created but remain empty, increasing index size
  - It may also be difficult to calculate the most appropriate size of the grid. In general, the rule of thumb is that the grid size should be approximately equal to the most common query window size

Indexing and Spatial Indexing

- The Quadtree
  - This index presents one solution to the distribution problem when using the Grid Structure
  - In this index type, the search space is decomposed into quadrants rather than equal-sized cells
  - The index is represented as a tree, with the root node having four leaf-nodes, and each leaf-node in turn having four further leaf-nodes as required by the data (hence Quadtree)

Indexing and Spatial Indexing

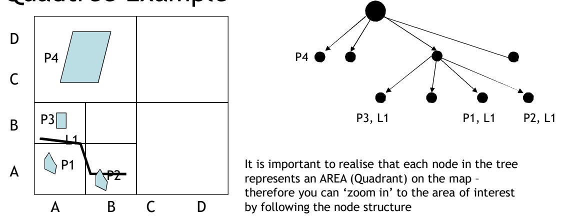
- The Quadtree
  - A line or a rectangle can appear in more than one leaf node.
  - Again each tree node is mapped onto a sub-area of the hard disk called a page
  - We will represent this as a tree structure

Indexing and Spatial Indexing

- The Quadtree
  - Note that as this is a space-based index, the depth of the tree varies depending on how densely populated the area of the map is
  - This may affect performance in densely populated areas

Indexing and Spatial Indexing

- Quadtree Example



It is important to realise that each node in the tree represents an AREA (Quadrant) on the map - therefore you can 'zoom in' to the area of interest by following the node structure

Assume that node order is:

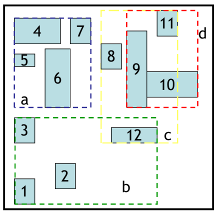
•Top Left, Top Right, Bottom Left, Bottom Right (this will depend on the software being used)

Indexing and Spatial Indexing

- R-Tree
  - This relies on a balanced hierarchical structure, in which each tree node is mapped onto a disk page
  - R-Trees organise rectangles according to a containment relationship
  - A rectangle (called the directory rectangle) is associated with each node.
  - This directory rectangle corresponds to the MBR of all the child rectangles or elements of this node

Indexing and Spatial Indexing

- R-Tree Directory Rectangles



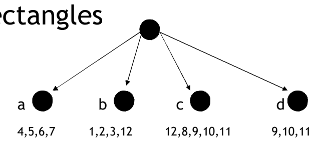


Diagram taken from RSV

Essentially, the R-Tree is based on the principle of looking at the data and taking into account what data is most likely to be queried at the same time

This then forms the basis of the index

Indexing and Spatial Indexing

- R-Tree
  - Can handle data in multiple dimensions
  - For all nodes in the tree except the root, the number of entries varies between 0 and ½ the total number of entries allowed on the node (this depends on disk page size)
  - For each entry in a non-leaf node, the entry is the directory rectangle of the child node
  - Each leaf entry contains the MBR of the object it links to
  - Each root has at least two entries (unless it is a leaf)
  - All leaves are at the same level

Indexing and Spatial Indexing

- R-Tree
  - The R-Tree adapts its gridding to the structure of the DATA rather than simply dividing up the search space
  - A region of search space populated with a large number of objects generates a large number of tree leaves
  - This allows the tree to have the same depth all through, giving equal performance for densely and non-densely populated areas

Indexing in PostGIS

- Creating indexes in PostgreSQL/PostGIS
  - Non-Spatial Index
    - Create Index spatial_table_points_name_idx ON spatial_table_points (name, surname);
  - Spatial Index
    - CREATE INDEX spatial_table_points_gidx ON spatial_table_points USING GIST(the_geom);

GIST stands for "Generalised Search Tree" which is a basic generic index that can be used for spatial and other data types. PostGIS then uses an R-Tree approach when implementing GIST on spatial datasets

Spatial Data Management

- Overview
  - Indexing
  - Normalisation and De-Normalisation
  - Query Parsing

Normalisation

- Normalisation
  - First rule of a database
    - "One fact, one place"

Normalisation

- Normalisation
  - Normalisation removes any redundant data from the database and avoids data duplication
  - It is a way of validating the quality of the database design
  - Is usually applied after the logical model has been developed

Normalisation

- Normalisation
  - A properly normalised set of tables simplifies the retrieval and maintenance processes
  - In an ideal world, we would not need to normalise the data, as the logical model would be perfect
  - But we need to go through the normalisation process to ensure that this is the case!

Normalisation

- Redundancies and Anomalies
  - A redundancy occurs when the same piece of data is duplicated in the database. This leads to excessive storage being used for the database.
  - Anomalies in the database related to problems with the following operations:
    - Update
    - Insert
    - Delete

Normalisation

- Update Anomaly
  - Data inconsistency or loss of data integrity can arise due to partial update (if data exists in two places, you could update only on instance)
- Insertion Anomaly
  - Data cannot be added because some other data is absent
- Deletion Anomaly
  - Data may be unintentionally lost through deletion of other data

Normalisation

- Normalisation
  - Reduces a table into simpler structure
  - Defined as a step-by-step process of transforming an non-normalised table with progressively simpler structures
  - Since the process is reversible, no information is lost during the transformation

– Decomposition

- Decomposition
  - This is the process of splitting up tables into smaller tables, which happens as part of the normalisation process
  - Should Be
    - Lossless – no information should be lost or added through the normalisation process, and the process should be reversible.
    - Preserve dependencies – the relationships between the different attributes and tables should not be lost.

– Normal Forms

- First Normal Form
  - For a table to be in First Normal Form (1NF) the underlying domains must contain simple atomic values
  - This means that there should only be one value in each field in the table

– First Normal Form

- Moving to First Normal Form – An Example

(this table is not Normalised)

Book ID	Book Title	Alternative Title	Authors	Series Title	Format	Font	Purchase Price
1	Database Systems	Concepts, Languages and Architectures	P. Atzeni S. Ceri S. Paraboschi R. Torlone				39.50

– First Normal Form

- Anomalies
  - Update Anomaly – you cannot modify an Author's name without having to re-insert values for the whole Authors field
  - Insert Anomaly – you cannot add a new Author without having to re-insert values for the whole Authors field
  - Delete Anomaly you cannot delete one Author without deleting the others as well

– First Normal Form

- Moving to First Normal Form

Book Title	Alternative Title	Authors	Publisher	Series Title	Format	Font	Cost Price
Database Systems	Concepts, Languages and Architectures	P. Atzeni S. Ceri S. Paraboschi R. Torlone	McGraw Hill	Database Tutor	Hardback	Times New Roman	20.00

Book Title	Alternative Title	Authors	Publisher	Format	Font	Series Title	Cost Price
Database Systems	Concepts, Languages and Architectures	P. Atzeni	McGraw Hill	Hardback	Times New Roman	Database Tutor	20.00
Database Systems	Concepts, Languages and Architectures	S. Ceri	McGraw Hill	Hardback	Times New Roman	Database Tutor	20.00
Database Systems	Concepts, Languages and Architectures	S. Paraboschi	McGraw Hill	Hardback	Times New Roman	Database Tutor	20.00
Database Systems	Concepts, Languages and Architectures	R. Torlone	McGraw Hill	Hardback	Times New Roman	Database Tutor	20.00

– First Normal Form

- First Normal Form
  - The above table has been normalised so that each field contains an atomic (single) value
  - This has created issues with duplicates, which are resolved by the next steps in the Normalisation process.

– Second Normal Form

- Moving to Second Normal Form – An Example

Book Title	Alternative Title	**Author**	Publisher	Format	Font	Series Title	Cost Price
Database Systems	Concepts, Languages and Architectures	P. Atzeni	McGraw Hill	Hardback	Times New Roman	Database Tutor	20.00
Database Systems	Concepts, Languages and Architectures	S. Ceri	McGraw Hill	Hardback	Times New Roman	Database Tutor	20.00
Database Systems	Concepts, Languages and Architectures	S. Paraboschi	McGraw Hill	Hardback	Times New Roman	Database Tutor	20.00
Database Systems	Concepts, Languages and Architectures	R. Torlone	McGraw Hill	Hardback	Times New Roman	Database Tutor	20.00
Database Design		M Jones	Bachmann	Paperback	Arial	Design for Dummies	15.00

(this table is in First Normal Form)

– Second Normal Form

- Anomalies
  - Update Anomaly – you cannot modify the Series Title without making sure that it is modified in four places
  - Insert Anomaly – you cannot add a new Author without having to re-insert values for the Book Title, Alternative Title and other fields as well
  - Delete Anomaly you cannot delete the ‘Database Design’ book without losing information regarding the publisher of the ‘Design for Dummies’ series

– Second Normal Form

- Second Normal Form
  - So, 1NF still shows anomalies for the update, insert and delete process
  - Therefore further normalisation is required to eliminate these
  - Second Normal Form (2NF) goes some way to addressing the problems shown by 1NF

– Second Normal Form

- Second Normal Form – An Example
  - Decompose the tables to eliminate these problems

– Second Normal Form

- Second Normal Form – An Example

Book Title	Author
Database Systems	S. Atzeni
Database Systems	S. Ceri
Database Systems	P. Paraboschi
Database Systems	R. Torlone
Database Design	M Jones

Book Title	Alternative Title	Cost Price	Series Title	Format	Font	Publisher
Database Systems	Concepts, Languages and Architectures	20.00	Database Tutor	Hardback	Times New Roman	McGraw Hill
Database Design		15.00	Design for Dummies	Paperback	Arial	Bachmann

Normalisation

- The Problem With Normalisation
  - It introduces “joins” into the database
  - Joins make a query run slower, especially when there is a lot of data in the tables
  - Indexes help, but may not be the whole solution on very large databases

Normalisation

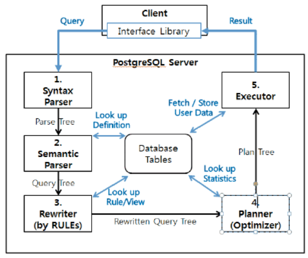
- De-Normalisation
  - Is the reverse process of normalisation
  - Tables are MERGED into one
  - This results in duplicate data
  - But … queries perform much faster as in a sense the joins between different parts of the data are “pre-calculated” so we don’t have to use JOIN statements

Spatial Data Management

- Overview
  - Indexing
  - Normalisation and De-Normalisation
  - Query Parsing

Query Parsing

- Examining Query Execution



Source: http://www.cubrid.org/blog/dev-platform/postgresql-at-a-glance/

Query Parsing

- Query Parsing – Step 1 – Syntax Parsing
  - Validates that the SQL syntax is correct
    - i.e. do the words 'from' and 'select' and so on appear as required.
  - Returns an error to the user if the syntax is incorrect

Source: http://files.meetup.com/1990051/PostgreSQL%20Internals%20-%20Overview.pdf

Query Parsing

- Query Parsing – Step 2 – Semantic Parsing
  - Validates that the SQL makes sense in the context of the database
    - i.e. do the named tables, views and columns exist
  - Returns an error to the user if the semantics are incorrect

http://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:2588723819082

Query Parsing

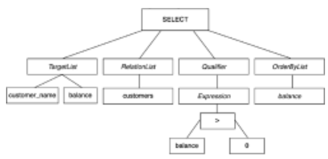
- Query Parsing – Step 3 – Query Rewriter
  - Modifies queries to take rules ("constraints") into consideration i.e. transforms the SQL into a form more appropriate for down-stream optimisation
    - e.g. translate name = 'Jack' or name = 'John' or name = 'Joe' to name in ('Jack', 'John', 'Joe') which is more efficient
    - e.g. if there is a sub query, check whether a join may be more efficient
    - e.g. re-order the query so that the where clause (filter) that eliminates most records is run first, leaving less records to be joineda
  - Once finished, the modified query is passed to the query planner for planning and execution – as a 'parse tree'

http://pic.dh e.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=%2Fcom.ibm.db2.luw.admin.perf.doc%2Fdoc%2Fc0005293.html
http://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:2588723819082

Query Parsing

- Query Parsing – Parse Tree

SELECT customer_name,
balance FROM customers
WHERE balance > 0
ORDER BY balance



Query Parsing

- Query Parsing – Step 4 – Query Optimization
  - The planner is responsible for traversing the parse tree and finding all possible plans for executing the query.
    - The plan might include a sequential scan through the entire table and index scans if useful indexes have been defined.
    - If the query involves two or more tables, the planner can suggest a number of different methods for joining the tables.

Query Parsing

- Query Parsing – Step 5 – Query Execution
  - The execution plans are developed in terms of query operators.
  - Each query operator transforms one or more input sets into an intermediate result set.
  - Complex queries are broken down into simple steps.
  - When all possible execution plans have been generated, the optimizer searches for the least-expensive plan.

Query Parsing

- Query Parsing – Step 5 – Operators (1)
  - Seq Scan (sequential scan).
    - Starts at the beginning of the table and scanning to the end of the table.
    - Checks each row against any 'where' clause and adds the row to the result if it passes

Query Parsing

- Query Parsing – Step 5 – Operators (2)
  - Index Scan
    - Traverses an index structure if there is a 'where' clause and an appropriate index exists
    - Allows the query to quickly skip any rows not meeting the criteria
    - Unlike the Seq Scan, returns the data pre-ordered (as the index is ordered)

Query Parsing

- Query Parsing – Step 5 – Operators (3)
  - Sort
    - Orders the result set returned by either the Seq Scan or Index operators
      - Data can be sorted in memory or on-disk (where temporary results are stored on disk if the system memory is not large enough for the sort) – the latter is slower.

Query Parsing

- Query Parsing – Step 5 – Operators (4)
  - The Unique operator eliminates duplicate values from the input set.
  - The LIMIT operator is used to limit the size of a result set.

Query Parsing

- Query Parsing – Step 5 – Query Execution
  - Taking the required operators and data into account, each plan is assigned an estimated execution cost.
  - Cost estimates are measured in units of disk I/O.
    - An operator that reads a single block of 8,192 bytes (8K) from the disk has a cost of one unit.
    - CPU time is also measured in disk I/O units, but usually as a fraction.
    - For example, the amount of CPU time required to process a single row is assumed to be 1/100th of a single disk I/O.

Query Parsing

- Query Parsing – Step 5 – Query Execution
  - After choosing the least-expensive execution plan, the query executor starts at the beginning of the plan and asks the topmost operator to produce a result set.
    - This in turn calls the next operator, which calls the next until the bottom most operator generates results which are passed back up the tree.

Query Parsing

- Query Parsing In Practice
- The EXPLAIN statement gives you some insight into how the PostgreSQL query planner/optimizer decides to execute a query.
  - The EXPLAIN statement can be used to analyze SELECT, INSERT, DELETE, UPDATE commands

Spatial Data Management

- Overview
  - Indexing
  - Normalisation and De-Normalisation
  - Query Parsing