WHERE ARE WE NOW?

W34: Introduction, DBMSs and Relational Databases
W35: Developing Database Systems
W36: SQL –Part I
W37: SQL –Part II and Relational Algebra
W38: Data Modelling
W39: Data Modelling
W40: Database Design
W41: Normalisation and Stored Procedures
W42: XML and Web Technology
W43: Processing XML Data
W44: XML Validation
W45: Beyond relational databases and XML
W46: File Organisations and Indexes
W47: Database Security and Administration
W48: Transaction Processing and Wrap-up

GOAL

- Yesterday:
  - *Database file organisation and access methods*
  - *Indexes and index file organisation*
  - *Managing Indexes in SQL*
- Today :
  - *Indexes and querying*
  - *MySQL engines and file organisation*
  - *Physical database design*

SAMPLE DATABASE (1)

- Employees db (https://launchpad.net/test-db/):
  - *9 departments*
  - *300 K employees*
  - *330 K employee affiliations*
  - *24 department head records*
  - *2.8 M salary records*
  - *440 K title records*

SAMPLE DATABASE (2)

- departments (dept_no, dept_name)
  - *PK dept_no*
- employees (emp_no, birth_date, first_name, last_name, gender, hire_date)
  - *PK emp_no*
- dept_emp (emp_no, dept_no, from_date, to_date)
  - *PK emp_no, dept_no*
  - *FK emp_no REFERENCES employees(emp_no)*
  - *FK dept_no REFERENCES departments (dept_no)*

SAMPLE DATABASE (3)

- dept_manager (emp_no, dept_no, from_date, to_date)
  - *PK emp_no, dept_no*
  - *FK emp_no REFERENCES employees(emp_no)*
  - *FK dept_no REFERENCES departments (dept_no)*
- salaries (emp_no, salary, from_date, to_date)
  - *PK emp_no, from_date*
  - *FK emp_no REFERENCES employees(emp_no)*
- titles (emp_no, title, from_date, to_date)
  - *PK emp_no, from_date*
  - *FK emp_no REFERENCES employees(emp_no)*

A NUT TO CRACK

- Write SQL statements for retrieving:
  - *The employee number and salary for those having salaries in the range of [70000, 75000>*
  - *The lowest salary ever*
  - *The number of employees currently receiving a salary*
  - *The number of employees currently working in each department*
  - *The employee number and names of employees having worked in more than two different departments*
  - *The average number of pay raises per employee*
  - *The names of all current employees of the Research department*
  - *The department name, name of the current manager, salary of the current manager, and the name and salaries of all employees of the department having a larger salary than the manager*
  - *The employee number and names of employees having the same name, ordered by name and employee number*
  - *The employee number and names of employees not currently receiving a salary*

INDEXES AND SELECTIONS

- Assumptions:
  - *4KB disk blocks*
  - *52B average record size*
  - *1 million records*
  - *3-level B+-tree unique index – root node cached*
- Database file size:
  - *1,000,000 * 52 B / 4096 B/block = 12,696 blocks*
- Sequential scan:
  - *Average reading length: 12,696/2 blocks = 6, 348 blocks*
- Indexed retrieval:
  - *3 blocks (2 index nodes + 1 database record)*

INDEXES AND JOINS

- Case:
  - *400,000 buildings – record size 72 B:*
    - » File size = 400,000 * 72 B / 4096 B/block = 703 blocks
  - *housing 3,200,000 technical installations in total (i.e., 8 installations per building on average) – record size 104 B:*
    - » File size = 3,200,000 * 104 B / 4096 B/block = 81,250 blocks
- Find joined info about 10 different buildings
  - *No index on foreign key:*
    - » Read entire technical installation file – possibly n times:
      - *n * 81,250 blocks*
  - *Index on foreign key:*
    - » Assuming indexes to same key are "clustered" but records not:
      - *2 * 10 + 80 blocks = 100 blocks*

QUERY PROCESSING AND QUERY OPTIMISATION

- Indexes are utilised by the query processor and query optimiser
- See C&B textbook Chapter 23 for more on query processing

FULL-TEXT INDEXES (1)

- Ordinary indexes do not work well on full-text:
  - `WHERE text LIKE '%word%'` *cannot make use of a regular index*
- Full-text indexes index individual terms in a text:
  - *Possibly except most common words*
  - *One entry for each term in the database:*
    - » Points to locations in texts in which the term appears
- Full-text query types:
  - *Boolean queries*
  - *Ranking queries*
  - *Ranking queries with query expansion*

FULL-TEXT INDEXES (2)

- Full-text indexes in MySQL:
  - *An index of type* `FULLTEXT`.
  - *Can be used only with MyISAM tables*
  - *Can be created only for* `CHAR`, `VARCHAR`, *or* `TEXT` *columns.*
  - *Full-text searching is performed using:*
    - » `MATCH() ... AGAINST` syntax.

MySQL ENGINES (1)

- MyISAM:
  - *The old MySQL storage engine*
- InnoDB:
  - *Provides transaction support*
  - *Supports FOREIGN KEY referential-integrity constraints.*
- Memory:
  - *Stores all data in RAM*
  - *Formerly known as the HEAP engine.*
- Archive:
  - *For storing and retrieving large amounts of seldom-referenced historical, archived, or security audit information.*
  - *INSERT and SELECT only operations supported*
- NDBCLUSTER (also known as NDB)
  - *Clustered database engine*
- ...

Feature	MyISAM	Memory	InnoDB	Archive	NDB
Storage limits	256TB	RAM	64TB	None	384EB
Transactions	No	No	Yes	No	Yes
Locking granularity	Table	Table	Row	Table	Row
MVCC	No	No	Yes	No	No
Geospatial data type support	Yes	No	Yes	Yes	Yes
Geospatial indexing support	Yes	No	No	No	No
B-tree indexes	Yes	Yes	Yes	No	Yes
Hash indexes	No	Yes	No [a]	No	Yes
Full-text search indexes	Yes	No	Yes [b]	No	No
Clustered indexes	No	No	Yes	No	No
Data caches	No	N/A	Yes	No	Yes
Index caches	Yes	N/A	Yes	No	Yes
Compressed data	Yes [c]	No	Yes [d]	Yes	No
Encrypted data [e]	Yes	Yes	Yes	Yes	Yes
Cluster database support	No	No	No	No	Yes
Replication support [f]	Yes	Yes	Yes	Yes	Yes
Foreign key support	No	No	Yes	No	No
Backup / point-in-time recovery [g]	Yes	Yes	Yes	Yes	Yes
Query cache support	Yes	Yes	Yes	Yes	Yes
Update statistics for data dictionary	Yes	Yes	Yes	Yes	Yes

A NUT TO CRACK

- What engine(s) to choose:
  - *Internet shop running online business*
  - *Historical data should be kept for archival purposes*

CONTENT

- *Indexes and querying*
- *MySQL engines and file organisation*
- *Physical database design:*
  - » Design steps
  - » Choosing file organisation and indexes
  - » Introducing controlled redundancy
- *NOSQL databases:*
  - » "Not only SQL" databases
- *JSON – an alternative to XML?*

PHYSICAL DATABASE DESIGN

- Steps:
  - *Translate logical data model for target DBMS*
    - » Design base relations
    - » Design representation of derived data
    - » Design general constraints
  - *Design file organisation and indexes*
    - » Analyse transactions
    - » Choose file organisations
    - » Choose indexes
    - » Estimate disk space requirements
  - *Design user views*
  - *Design security mechanisms*
  - *Consider controlled redundancy*
  - *Monitor and tune the operational system*

CHOOSING FILE ORGANISATION

- In many cases, a relational DBMS may give little or no choice for choosing file organisation
- We saw MySQL offer a few engines using different file organisation:
  - *MyISAM*
  - *InnoDB*
  - *Memory*
  - *Archive*
  - *NDBCLUSTER (also known as NDB)*

CHOOSING INDEXES (1)

- Advantages of having indexes:
  - *Fast retrieval in large files*
- Disadvantages:
  - *Slower inserts and deletes*
  - *Storage space needed for index file*
  - *The query optimiser may need more time to optimise*
- Therefore:
  - *Do not add indexes "by default"*

CHOOSING INDEXES (2)

- When to add an index:
  - *Add index on frequently used foreign keys*
  - *Add index on attributes that are heavily used as secondary keys*
  - *Add index on attributes that are frequently involved in:*
    - » selection/join, `ORDER BY`, `DISTINCT`, ...
  - *Add index to attributes involved in aggregate functions:*
    - » `MIN, MAX`
- When not to index:
  - *Small relations*
  - *Attributes/relations are frequently updated*
  - *Attributes in queries retrieving a significant portion of records*
  - *Attributes consist of long character strings*

INTRODUCING CONTROLLED REDUNDANCY

- Normalisation good for:
  - *Complexity*
  - *Flexibility*
  - *Speed of updates*
- But sometimes the query performance suffers
- When to consider denormalisation (C&B 19):
  - *Performance is unsatisfactory*
  - *Relation has a low update rate*
  - *Relation has a very high query rate*