Useful reference: Microsoft Developer Network Library (http://msdn.microsoft.com/library). Drill down to Servers and Enterprise Development → SQL Server → SQL Server 2000 → SDK Documentation → Creating and Using Data Warehouses

Basic material

- ✓ Order of evaluation of a basic SQL statement: SELECT ③… FROM ①… WHERE ②…
- ✓ Views help to simplify queries, and are good for security too, since the query may yield different results for different users.
- ✓ Indexes may be explicitly created to improve performance, but bear in mind that sequential scans *may* be quicker for some queries!
- ✓ Cluster indexes (as used by Oracle) take up space; hash indexes (using a hash function) do not. Cluster indexes can be used for partial matches; hash indexes can not.

Data Warehousing

The purpose of Data Warehouses (DWs) are to enable knowledge workers to make faster and better informed decisions. With this in mind, a DW can be described as being a copy of transaction data specifically structured for querying and reporting (Kimball) or a non-volatile collection of data used primarily in decision making (Inmon). A Data Warehouse is a repository of data for the use of querying, analysis and reporting (me!).

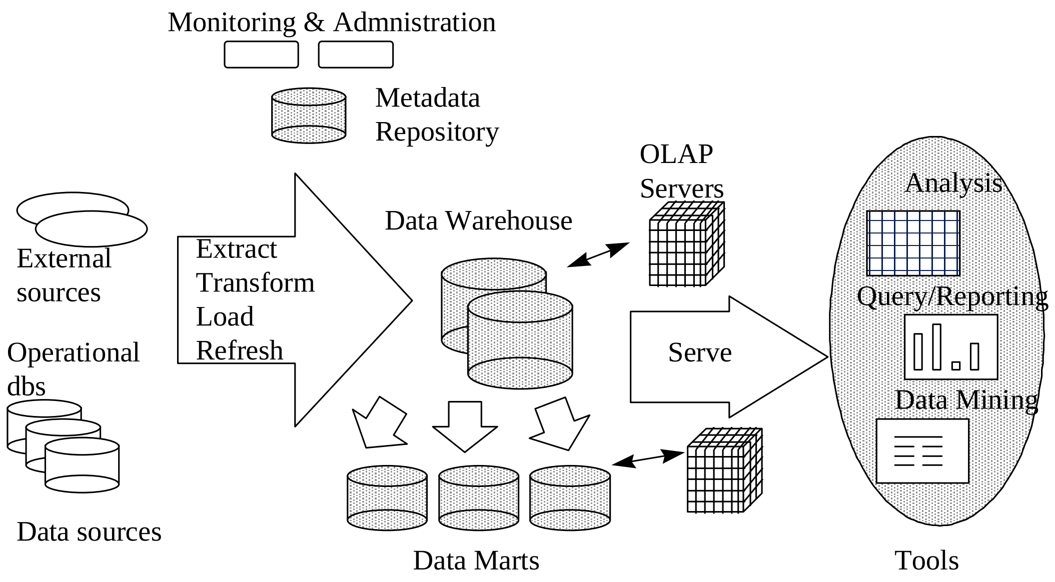
Key issues to consider: **feasibility** (do we need a DW?), **requirements** (what data do we need to store?), **design** (how best to organise the data?), **implementation** (when & how will we collect the data?) and **maintenance** (when & how will updates be propagated?).

How do DW & OLAP differ from an OLTP database?

Database & OLTP used for core business, for reliable and efficient processing of large number of transactions whilst accessing few records (e.g. banks, stock exchanges, Amazon) whereas DW & OLAP used for decision support, processing complex queries not always known in advance, accessing large number of records drawn from disparate data sources, for purposes of visualising trends, etc, or for finding meaning in inherently meaningless information (e.g. SETI). Analysts want to discover trends and anomalies in data – OLAP is a tool designed to facilitate this kind of analysis. OLTP and pre-defined reports and queries are not suited to this explorative analysis.

From *Chaudhari & Dayal*: "OLTP applications typically automate clerical data processing tasks." "Data warehouses are targeted for *decision support*." "Data in a warehouse is typically modelled *multidimensionally*." DWs implemented on RDBMSs are called Relational OLAP (**ROLAP**) servers. Multidimensional OLAP (**MOLAP**) servers directly store multidimensional data in special data structures [Hybrid OLAP – **HOLAP** – uses a combination]. "Data Marts are departmental subsets focused on selected subjects (e.g. a marketing data mart may include customer, product, and sales information)."

Memorise this diagram!



ETL (left-hand side of diagram)
Often a large proportion of time is spent extracting data, cleansing it and loading (aka ETL – extract, transform, load).

Problems with data: **Incomplete** (e.g. missing records), **Incorrect** (duplicates, wrong aggregations), **Incomprehensible** (unknown data, multiple fields within one field), **Inconsistent** (lack of referential integrity, different timings, use of codes).

There are three different classes of data cleaning tools: *migration* tools, *scrubbing* tools and *auditing* tools.

When **loading** a DW, massive amounts of data have to be moved, often in a small time window when the DW is offline. This can be speeded by moving blocks of data *en masse* rather than record-by-record, and/or by turning off transaction logging and rollback features. For continual access to a DW, new data might be added incrementally, or in parallel with a quick 'swap' at the end (rather as operating systems perform copying of large files: copy to a temporary file, then rename file as the last – and quick – operation).

For **refreshing** a DW there are three considerations: *when* to refresh, *how* to refresh and *what* to refresh. Unless there is a need for current data (e.g. stock ticker prices) then days can be refreshed periodically (see **loading** above).

OLAP (right-hand side of diagram)
Popular conceptual model for front-end tools is the **multidimensional** view of data in a DW. Another distinctive feature of the conceptual model for OLAP is *aggregation*, that is, the rolling up of data along certain dimensions. **Pivot tables** are an example of the realisation of this. To be suitable for such viewing, data should include duplicate values in one or more fields (columns) include data suitable for aggregation (e.g. numeric). OLAP operators related to pivoting are **rollup** and **cube** (aggregation of measure values, order is important in *rollup* but not in *cube*) and its converse **drill-down**.

Slice and **dice** operations refer to reducing the dimensionality of the data, i.e. producing subsets on which to operate (*slice* = select on one dimension, *dice* = select on multiple dimensions). OLAP's multidimensional data model and data aggregation techniques organize and summarise large amounts of data so it can be evaluated quickly using online analysis and graphical tools.

SQL Extensions that facilitate OLAP queries include multiple group-by functions such as *rollup* and *cube* (SQL99), and aggregate functions such as *rank* and moving average functions such as *window*, as well as statistical functions such as *mean*, *mode* and *median* (all SQL2000).

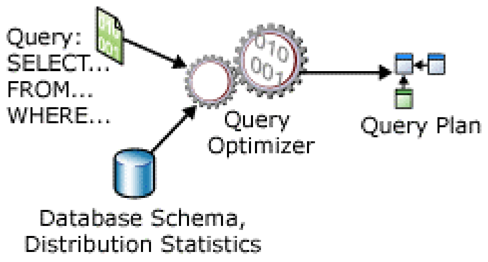
Remember that rollup and cube introduce records with NULL values. Remember too that there are NULL values and there are *truly* NULL values! Usually there is a way to differentiate between these.

Metadata (top side of diagram)
Metadata is data about the data in the warehouse. Three categories: *(i) Administrative metadata* includes metadata about both data and data processing. The former includes schemas of source databases, back-end and front-end tools, definitions of DW schema, dimensions and hierarchies, predefined queries and reports, data mart and partitioning details. The latter includes rules for ETL, refresh and purging polices, as well as access control details and policies; *(ii) Business metadata* includes definitions and terms, and charging policies; *(iii) Operational metadata* includes descriptions of active information such usage reports and audit trails.

Creating and managing a DW is difficult and a range of development tools are available to assist with developing DW schemas, views, scripts, queries, reports, etc, as well as monitoring the usage and performance of the DW.

Logical design:
- ✓ consider **multidimensional** model,
- ✓ **measures** (e.g. numeric values), classified as additive, semi- or non-additive,
- ✓ **dimensions** (e.g. time, customer, product),
- ✓ **attributes** (e.g. customer number, name, address),
- ✓ **fact tables** (star hub), contain information regarding *measures*,
- ✓ **dimension tables** (star extremities), often hierarchical (e.g. time – year/fiscal period, quarter, month, week, day, or store – country, region, city, id)
- ✓ **Materialised views** are an important technique for maximising the query performance of any decision support system [Oracle]. The benefits are threefold: (1) no need to join tables with every query saving processing costs, (2) queries are simpler saving development costs, (3) queries do not need modification if the underlying data is updated (the materialised view is updated instead).
- ✓ Most DWs use a **star schema** to represent the multidimensional data model. **Snowflake schemas** provide a refinement of star schemas. *A snowflake schema is a normalised star schema.* Normalising reduces redundancy but is less speed-efficient – more joins needed. Schemas with a number of fact tables (i.e. star schemas) are often known as **fact constellations**!

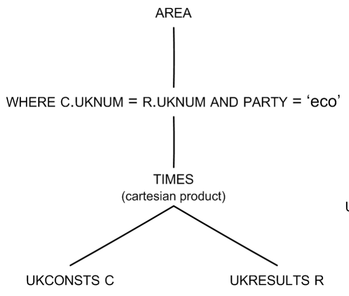
Query optimisation: Query trees & Query plans
query optimisation is a big subject area, and particularly relevant for DWs where queries can be quite complicated and operate over huge tables



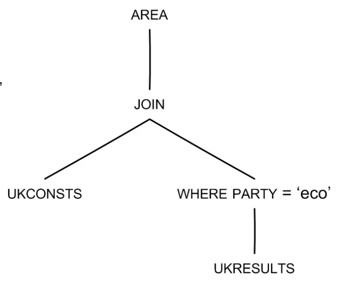
A **query tree** corresponds to an expression where the input relations are represented as leaf nodes and relational operators as internal nodes. There are many possible trees for a given query, e.g.

SELECT AREA
FROM UKCONSTS C, UKRESULTS R
WHERE C.UKNUM = R.UKNUM
AND PARTY = 'eco'

can be illustrated as either


AREA
WHERE C.UKNUM = R.UKNUM AND PARTY = 'eco'
TIMES (cartesian product)
UKCONSTS C
UKRESULTS R

or



Reducing the sizes of partial results early in the tree generally makes for a more optimum execution cost. The most optimal query tree is that with the estimated lowest execution cost, which may relate to processing time or disk accesses, or a combination.

Query plans: The query optimiser produces a series of plans using different algorithms and trees, and chooses the one with minimal estimated cost. It also uses database statistics and heuristics to determine the optimal plan. The best plan can use either a *rules-based* or a *cost-based* approach (SQL Server and Oracle use the cost-based approach). Rules-based is more rigid; cost-based is more advanced. One goal is best-throughput which can either be best time to completion or best response time for first record(s), depending on the application.
Optimisers can be influenced by setting parameters (e.g. OPTIMIZER_GOAL) or by giving optimiser hints in SQL statements via specially formatted comments
(e.g. `SELECT /*+ FIRST_ROWS */ …`).

Physical design:
Mostly performance-vs-space-vs-cost-vs-maintenance related. Consider **partitioning** or **striping** (breaking large files or tables into smaller ones), **parallelism** (allowing concurrent access to data by different processes), **bit map indices**.

DWs contain large volumes of data. Efficient processing of this data require extra structures such as **indices** and **materialised views**. Choosing which of these to build is am important physical design problem. Optimisation and exploiting parallelism of complex queries is also an important design area.

DW servers can use **bit map indices** for speed efficiency (in addition to b-tree indexes), although these can be costly in terms of space and effort required for updating (therefore best for non-volatile data). They can speed up index operations such as intersection, union, join and aggregation.

The selection of views to materialise should take into account the costs of updating, the storage needs for such views, but mostly the types of queries asked of the data.

Different **server architectures** can be employed, such as siting ROLAP servers between a relational back-end and the client front-end tools. MOLAP servers directly support the multidimensional view of data desired by clients. Specialised SQL servers can offer enhanced SQL directly to clients. HOLAP can use vertical partitioning to store aggregations in MOLAP for fast access and detailed data in ROLAP to optimize cube processing, or it can employ horizontal partitioning to store recent, frequently accessed data in MOLAP and older or less-used data in ROLAP.

Performance: consider schemas, views, partitioning, parallelism, optimising queries, etc.

<u>Data Mining (DM)</u> contrasted with OLAP:
OLAP organises data into multidimensional structures to facilitate exploration; DM is the process of performing the explorative analysis. The results of DM can be used with OLAP to improve explorative analysis, for example, DM may find groups of customers according to particular attributes; these 'new' groups can be used by OLAP to create new dimensions to allow DM to explore the data from this new perspective.

OLAP uses multidimensional data representations, call cubes, to provide rapid access to DW data. DM uses algorithms to analyze data and create models that represent information about the data. *OLAP creates cubes, DM creates models.*

Generally, OLAP is used to tell you *what* happened, and data mining is used to tell you *why* (knowledge discovery). This example is from retail banking:

> "We used OLAP to tell us which accounts were in arrears, broken down by geography, type of loan, customer segment, interest rate and so on. We could 'drill into' and 'slice and dice' our arrears cube to identify which part of our loan book had the most arrears.
>
> "We used data mining to help us predict in advance whether a loan was likely to go into arrears. To do this, we identified some key attributes for our borrowers (age, years with bank, income, number of other accounts etc) and got hold of some data that described how the loans had performed since they were taken out. We then used a data mining model to build up a scoring system that, given a certain set of borrower attributes and a previous history of payments, would predict whether the loan was likely to default in the next few months.
>
> "In summary, OLAP told us which loans were defaulting and how they broke down, whilst data mining allowed us to build up a model that predicted future performance of the loan, given a certain type of borrower and a certain set of circumstances."

Most of the DM effort is expended on creating the target data and choosing the DM algorithm. The actual data mining is often the easiest task!

Reasons for performing DM include prediction, classification, clustering, link (association) analysis.

DM (Knowledge Discovery) techniques include association rule mining, decision trees, K-means clustering, neural networks.

Association rule mining searches for interesting relationships (association rules) among data. Association rules are identified by checking that they exhibit appropriate **confidence** and **support**. Both are expressed as ratios (≤ 1) or percentages.
Confidence for a rule $A \rightarrow B$ is defined as *(records containing A & B) / (records containing A)* or *if A, what chance of B?*
Support for a rule $A \rightarrow B$ is defined as (*records containing A & B) / (total records)* or *what chance of A & B?*
Support tends to be lower than confidence. e.g. the rule "people who buy cereal also buy milk" may have support 5% (i.e. only 1 in 20 buys cereal and milk) and confidence 75% (i.e. 3 out of 4 people who buy cereal also buy milk). This can be expressed as "Cereal→Milk (support 5%, confidence 75%)".

Apriori is an efficient association rule mining algorithm used to find all sets of items that have at least a minimum level of support.
It works as follows (1) from records of transactions in DB generate all 1-itemsets (i.e. with one item within it); (2) eliminate itemsets which do not have the necessary support (i.e. which don't occur often enough); (3) with the remaining itemsets, generate all possible 2-itemsets; (4) repeat from (2) until no more *k*-itemsets can be found.

Relationships which exceed both required minimum support *and* confidence thresholds are called **strong rules**. Note that not all strong rules are believable – some need further analysis, some may be coincidental (e.g. people who buy cereal may also buy ketchup, but the two are – hopefully – not linked!).

Apriori can be improved by reducing the number of passes over the entire dataset (i.e. the number of accesses)

Auto Med (*auto*matic *med*iation)
A framework and software package based on a data integration approach called **Both-as-view** (BAV). This can be used to derive both **Local-as-view** (LAV) and **Global-as-view** (GAV). Can integrate DBMSs, XML files and structured flat files. By using a Hypergraph Data Model (**HDM**) AutoMed is capable of automating the translation between these data models.

In AutoMed, schemas are incrementally transformed by applying to them a sequence of primitive transformations. Each transformation is accompanied by a query expressing in AutoMed's Intermediate Query Language (**IQL**). For each transformation there is an automatically derivable *reverse transformation*. AutoMed stores metadata about both data and data processing. The former includes the schemas of the data sources, warehouse and data marts, ownership of the data, etc. The latter includes rules for data ETL, refresh policies, etc.

Using AutoMed, four steps are needed in order to create the metadata expressing the DW schemas and transformations: (1) create AutoMed repositories, (2) specify data models, (3) extract data source schemas, (4) define transformation pathways.

AutoMed differs from a conventional DW approach using a single conceptual data model (**CDM**) in three main ways: (1) data source schemas as translated into AutoMed representations without loss of information (CDMs can suffer from semantic mismatches); (2) independence – decoupling – from any particular CDM; (3) changed to data source schemas easily incorporated.

Example queries

Example moving average query using 'window':

```
SELECT SM.S#, SM.MONTH, SUM(QUANTITY)
           OVER W AS "3 SUPPLY TOTAL"
FROM SUPPLY MONTHLY SM
WINDOW AS (PARTITION BY SM.S#
                  ORDER BY SM.MONTH
                  RANGE 2 PRECEDING);
```

Example "top *n*" query using 'rank':

```
SELECT SALES_AND_RANK.STORE_ID, SALES_AND_RANK.TOTAL_SALES
FROM (SELECT SALES.STORE_ID, SALES.TOTAL_SALES,
                     DENSE_RANK() OVER (ORDER BY SALES.TOTAL_SALES DESC)
                     AS TOTAL_SALES_RANK
          FROM SALES) SALES_AND_RANK
WHERE TOTAL_SALES_RANK <= 10;
```