structures such as priority queues and sets. Because it keeps all data in memory, its implementation is comparatively simple.

Recent research indicates that an in-memory database architecture could be extended to support datasets larger than the available memory, without bringing back the overheads of a disk-centric architecture [45]. The so-called *anti-caching* approach works by evicting the least recently used data from memory to disk when there is not enough memory, and loading it back into memory when it is accessed again in the future. This is similar to what operating systems do with virtual memory and swap files, but the database can manage memory more efficiently than the OS, as it can work at the granularity of individual records rather than entire memory pages. This approach still requires indexes to fit entirely in memory, though (like the Bitcask example at the beginning of the chapter).

Further changes to storage engine design will probably be needed if *non-volatile memory* (NVM) technologies become more widely adopted [46]. At present, this is a new area of research, but it is worth keeping an eye on in the future.

Transaction Processing or Analytics?

In the early days of business data processing, a write to the database typically corresponded to a *commercial transaction* taking place: making a sale, placing an order with a supplier, paying an employee’s salary, etc. As databases expanded into areas that didn’t involve money changing hands, the term *transaction* nevertheless stuck, referring to a group of reads and writes that form a logical unit.

> **NOTE**
>
> A transaction needn't necessarily have ACID (atomicity, consistency, isolation, and durability) properties. *Transaction processing* just means allowing clients to make low-latency reads and writes—as opposed to *batch processing* jobs, which only run periodically (for example, once per day). We discuss the ACID properties in Chapter 7 and batch processing in Chapter 10.

Even though databases started being used for many different kinds of data—comments on blog posts, actions in a game, contacts in an address book, etc.—the basic access pattern remained similar to processing business transactions. An application typically looks up a small number of records by some key, using an index. Records are inserted or updated based on the user's input. Because these applications are interactive, the access pattern became known as *online transaction processing* (OLTP).

However, databases also started being increasingly used for *data analytics*, which has very different access patterns. Usually an analytic query needs to scan over a huge number of records, only reading a few columns per record, and calculates aggregate statistics (such as count, sum, or average) rather than returning the raw data to the user. For example, if your data is a table of sales transactions, then analytic queries might be:

- What was the total revenue of each of our stores in January?
- How many more bananas than usual did we sell during our latest promotion?
- Which brand of baby food is most often purchased together with brand X diapers?

These queries are often written by business analysts, and feed into reports that help the management of a company make better decisions (*business intelligence*). In order to differentiate this pattern of using databases from transaction processing, it has been called *online analytic processing* (OLAP) [47].[iv] The difference between OLTP and OLAP is not always clear-cut, but some typical characteristics are listed in Table 3-1.

Table 3-1. Comparing characteristics of transaction processing versus analytic systems

Property	Transaction processing systems (OLTP)	Analytic systems (OLAP)
Main read pattern	Small number of records per query, fetched by key	Aggregate over large number of records
Main write pattern	Random-access, low-latency writes from user input	Bulk import (ETL) or event stream
Primarily used by	End user/customer, via web application	Internal analyst, for decision support
What data represents	Latest state of data (current point in time)	History of events that happened over time
Dataset size	Gigabytes to terabytes	Terabytes to petabytes

At first, the same databases were used for both transaction processing and analytic queries. SQL turned out to be quite flexible in this regard: it works well for OLTP-type queries as well as OLAP-type queries. Nevertheless, in the late 1980s and early 1990s, there was a trend for companies to stop using their OLTP systems for analytics purposes, and to run the analytics on a separate database instead. This separate database was called a *data warehouse*.

Data Warehousing

An enterprise may have dozens of different transaction processing systems: systems powering the customer-facing website, controlling point of sale (checkout) systems in physical stores, tracking inventory in warehouses, planning routes for vehicles, managing suppliers, administering employees, etc. Each of these systems is complex and needs a team of people to maintain it, so the systems end up operating mostly autonomously from each other.

These OLTP systems are usually expected to be highly available and to process transactions with low latency, since they are often critical to the operation of the business. Database administrators therefore closely guard their OLTP databases. They are usually reluctant to let business analysts run ad hoc analytic queries on an OLTP database, since those queries are often expensive, scanning large parts of the dataset, which can harm the performance of concurrently executing transactions.

A *data warehouse*, by contrast, is a separate database that analysts can query to their hearts' content, without affecting OLTP operations [48]. The data warehouse contains a read-only copy of the data in all the various OLTP systems in the company. Data is extracted from OLTP databases (using either a periodic data dump or a continuous stream of updates), transformed into an analysis-friendly schema, cleaned up, and then loaded into the data warehouse. This process of getting data into the warehouse is known as *Extract–Transform–Load* (ETL) and is illustrated in Figure 3-8.

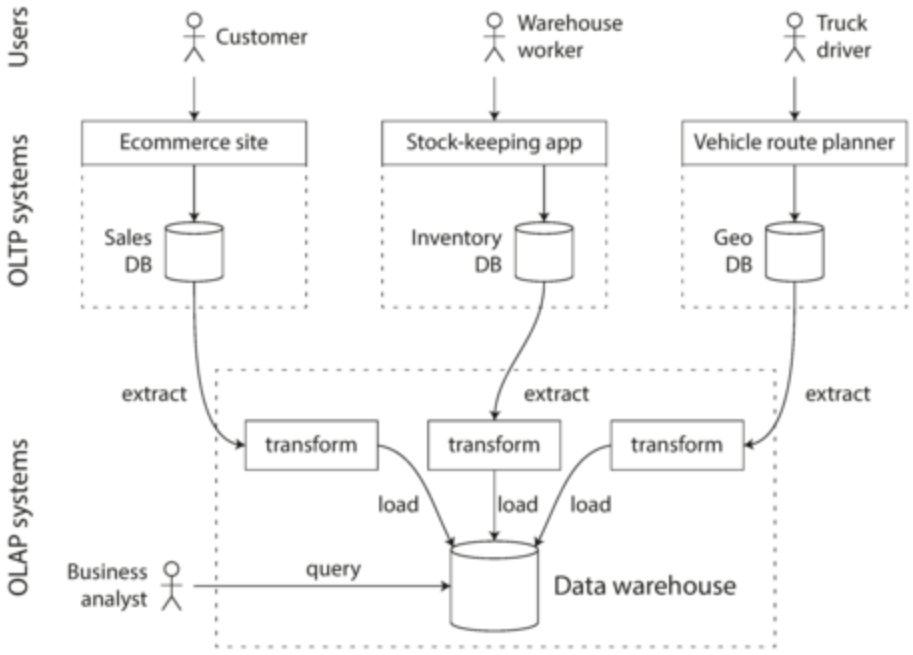

Figure 3-8. Simplified outline of ETL into a data warehouse.

Data warehouses now exist in almost all large enterprises, but in small companies they are almost unheard of. This is probably because most small companies don't have so many different OLTP systems, and most small

companies have a small amount of data—small enough that it can be queried in a conventional SQL database, or even analyzed in a spreadsheet. In a large company, a lot of heavy lifting is required to do something that is simple in a small company.

A big advantage of using a separate data warehouse, rather than querying OLTP systems directly for analytics, is that the data warehouse can be optimized for analytic access patterns. It turns out that the indexing algorithms discussed in the first half of this chapter work well for OLTP, but are not very good at answering analytic queries. In the rest of this chapter we will look at storage engines that are optimized for analytics instead.

The divergence between OLTP databases and data warehouses

The data model of a data warehouse is most commonly relational, because SQL is generally a good fit for analytic queries. There are many graphical data analysis tools that generate SQL queries, visualize the results, and allow analysts to explore the data (through operations such as *drill-down* and *slicing and dicing*).

On the surface, a data warehouse and a relational OLTP database look similar, because they both have a SQL query interface. However, the internals of the systems can look quite different, because they are optimized for very different query patterns. Many database vendors now focus on supporting either transaction processing or analytics workloads, but not both.

Some databases, such as Microsoft SQL Server and SAP HANA, have support for transaction processing and data warehousing in the same product. However, they are increasingly becoming two separate storage and query engines, which happen to be accessible through a common SQL interface [49, 50, 51].

Data warehouse vendors such as Teradata, Vertica, SAP HANA, and ParAccel typically sell their systems under expensive commercial licenses. Amazon RedShift is a hosted version of ParAccel. More recently, a plethora of open source SQL-on-Hadoop projects have emerged; they are young but

aiming to compete with commercial data warehouse systems. These include Apache Hive, Spark SQL, Cloudera Impala, Facebook Presto, Apache Tajo, and Apache Drill [52, 53]. Some of them are based on ideas from Google's Dremel [54].

Stars and Snowflakes: Schemas for Analytics

As explored in Chapter 2, a wide range of different data models are used in the realm of transaction processing, depending on the needs of the application. On the other hand, in analytics, there is much less diversity of data models. Many data warehouses are used in a fairly formulaic style, known as a *star schema* (also known as *dimensional modeling* [55]).

The example schema in Figure 3-9 shows a data warehouse that might be found at a grocery retailer. At the center of the schema is a so-called *fact table* (in this example, it is called `fact_sales`). Each row of the fact table represents an event that occurred at a particular time (here, each row represents a customer's purchase of a product). If we were analyzing website traffic rather than retail sales, each row might represent a page view or a click by a user.

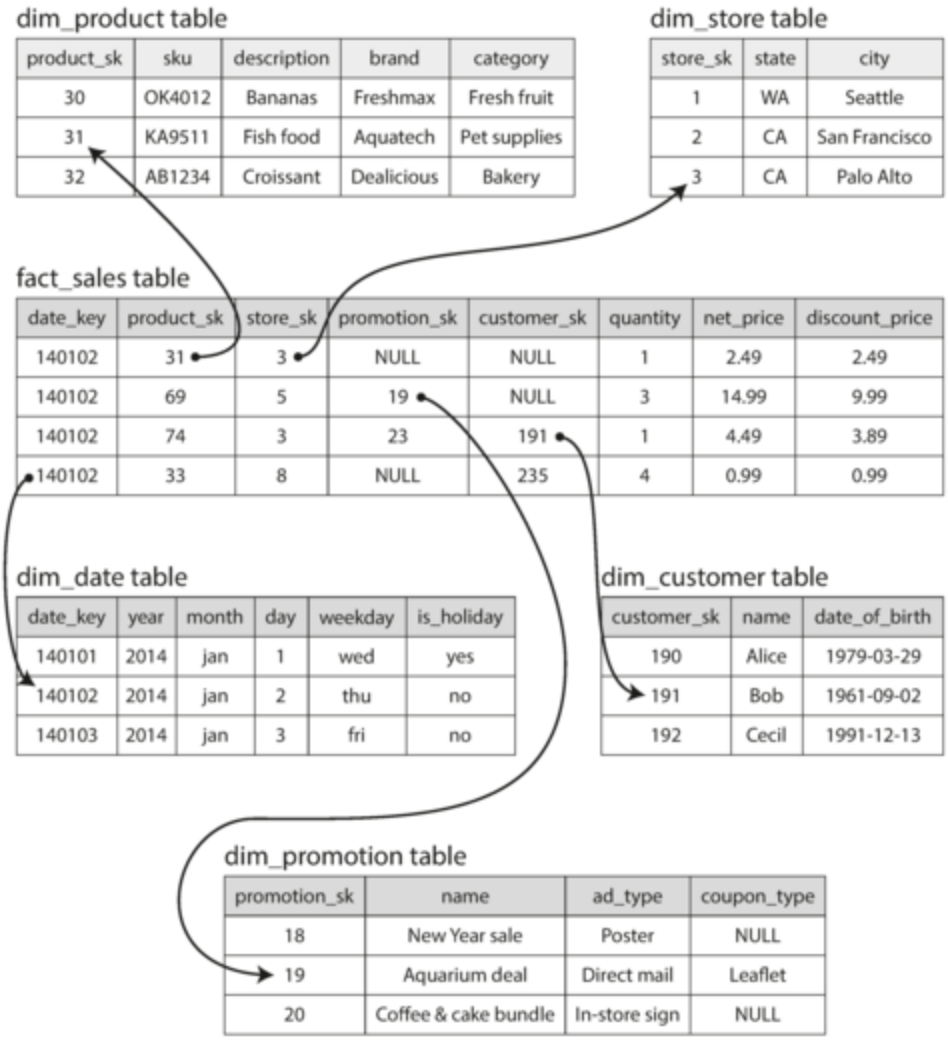

Figure 3-9. Example of a star schema for use in a data warehouse.

Usually, facts are captured as individual events, because this allows maximum flexibility of analysis later. However, this means that the fact table can become extremely large. A big enterprise like Apple, Walmart, or eBay may have tens of petabytes of transaction history in its data warehouse, most of which is in fact tables [56].

Some of the columns in the fact table are attributes, such as the price at which the product was sold and the cost of buying it from the supplier (allowing the profit margin to be calculated). Other columns in the fact table are foreign key references to other tables, called *dimension tables*. As each row in the fact table represents an event, the dimensions represent the *who*, *what*, *where*, *when*, *how*, and *why* of the event.

For example, in Figure 3-9, one of the dimensions is the product that was sold. Each row in the `dim_product` table represents one type of product that is for sale, including its stock-keeping unit (SKU), description, brand name, category, fat content, package size, etc. Each row in the `fact_sales` table uses a foreign key to indicate which product was sold in that particular transaction. (For simplicity, if the customer buys several different products at once, they are represented as separate rows in the fact table.)

Even date and time are often represented using dimension tables, because this allows additional information about dates (such as public holidays) to be encoded, allowing queries to differentiate between sales on holidays and non-holidays.

The name "star schema" comes from the fact that when the table relationships are visualized, the fact table is in the middle, surrounded by its dimension tables; the connections to these tables are like the rays of a star.

A variation of this template is known as the *snowflake schema*, where dimensions are further broken down into subdimensions. For example, there could be separate tables for brands and product categories, and each row in the `dim_product` table could reference the brand and category as foreign keys, rather than storing them as strings in the `dim_product` table. Snowflake schemas are more normalized than star schemas, but star schemas are often preferred because they are simpler for analysts to work with [55].

In a typical data warehouse, tables are often very wide: fact tables often have over 100 columns, sometimes several hundred [51]. Dimension tables can also be very wide, as they include all the metadata that may be relevant for analysis—for example, the `dim_store` table may include details of which services are offered at each store, whether it has an in-store bakery, the square footage, the date when the store was first opened, when it was last remodeled, how far it is from the nearest highway, etc.

Column-Oriented Storage