

HCL Informix 14.10

Data warehousing



Contents

- Chapter 1. Data warehousing..... 1**
 - Dimensional databases..... 1
 - Dimensional databases..... 1
 - Design a dimensional data model..... 7
 - Implement a dimensional database..... 34
 - Performance tuning dimensional databases..... 44
 - Informix® Warehouse Accelerator..... 52
 - Overview of..... 52
 - Installation..... 70
 - Configuration..... 76
 - Create an accelerator..... 94
 - Data marts and AQTs..... 95
 - Turning on query acceleration..... 129
 - SQL administration routines..... 134
 - Informix® Warehouse Accelerator.....
 - Troubleshooting..... 170
 - Sample warehouse schema..... 171
 - Sysmaster interface (SMI) pseudo tables for query probing data..... 174
- Index..... 177**

Chapter 1. Data warehousing

In addition to designing and implementing Informix® dimensional databases, you can use tools to create data warehouse applications and optimize your data warehouse queries.

Dimensional databases

The *Informix® Data Warehouse Guide* provides information to help you design, implement, and manage dimensional databases, and describes the tools that you can use to create data warehouses and optimize your data warehouse queries.

These topics are of interest to the following users:

- Database administrators
- System administrators
- Performance engineers

These topics are written with the assumption that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with dimensional databases, relational databases, or exposure to database concepts
- Some experience with database server administration, operating-system administration, or network administration

Dimensional databases

A dimensional database is a relational database that uses a *dimensional data model* to organize data. This model uses fact tables and dimension tables in a star or snowflake schema.

A dimensional database is the optimal type of database for data warehousing.

The availability and reliability of the Informix® database server includes a full active-active cluster solution for high availability and low cost scalability. You can use Informix® to manage workload distribution across multiple read-only or full-transaction nodes. You can dynamically add different types of nodes into your cluster environment to scale out or increase availability in the most demanding environments.

Warehouse workloads have the flexibility to work on the same database with operational data, running real-time on a separate node in the cluster. Data can also be replicated in real-time using Enterprise Replication, or copied to a separate data warehouse server. With Informix®, you have the flexibility to design the system to meet your needs and to make the most of your existing infrastructure.

Overview of data warehousing

Data warehouse databases provide a decision support system (DSS) environment in which you can evaluate the performance of an entire enterprise over time.

In the broadest sense, the term *data warehouse* is used to refer to a database that contains very large stores of historical data. The data is stored as a series of snapshots, in which each record represents data at a specific time. By analyzing these snapshots you can make comparisons between different time periods. You can then use these comparisons to help make important business decisions.

Data warehouse databases are optimized for data retrieval. The duplication or grouping of data, referred to as *database denormalization*, increases query performance and is a natural outcome of the dimensional design of the data warehouse. By contrast, traditional online transaction processing (OLTP) databases automate day-to-day transactional operations. OLTP databases are optimized for data storage and strive to eliminate data duplication. Databases that achieve this goal are referred to as *normalized* databases.

An enterprise data warehouse (EDW) is a data warehouse that services the entire enterprise. An *enterprise data warehousing environment* can consist of an EDW, an operational data store (ODS), and physical and virtual data marts.

A data warehouse can be implemented in several different ways. You can use a single data management system, such as Informix®, for both transaction processing and business analytics. Or, depending on your system workload requirements, you can build a data warehousing environment that is separate from your transactional processing environment.

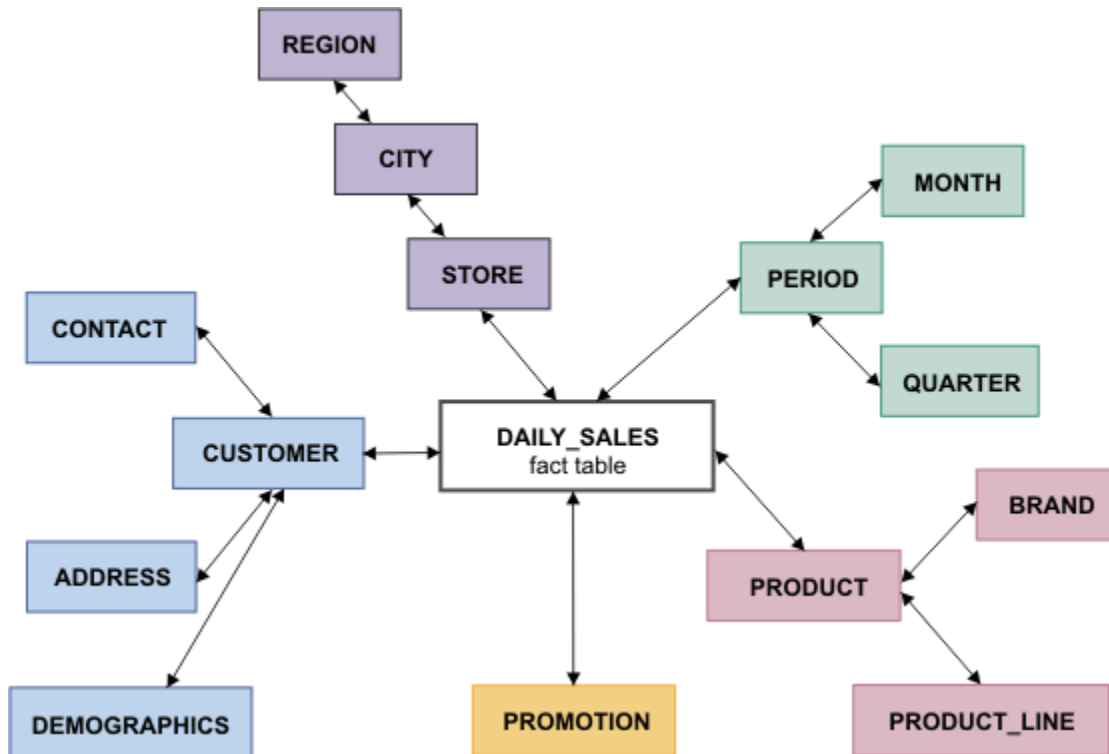
Informix® uses the umbrella terms *data warehousing* and *data warehousing environment* to encompass any of the following forms that you might use to store your data:

Data warehouse

A database that is optimized for data retrieval to facilitate reporting and analysis. A data warehouse incorporates information about many subject areas, often the entire enterprise. Typically you use a dimensional data model to design a data warehouse. The data is organized into dimension tables and fact tables using star and snowflake schemas. The data is denormalized to improve query performance. The design of a data warehouse often starts from an analysis of what data already exists and how to collect it in such a way that the data can later be used. Instead of loading transactional data directly into a warehouse, the data is often integrated and transformed before it is loaded into the warehouse.

The primary advantage of a data warehouse is that it provides easy access to and analysis of vast stores of information on many subject areas.

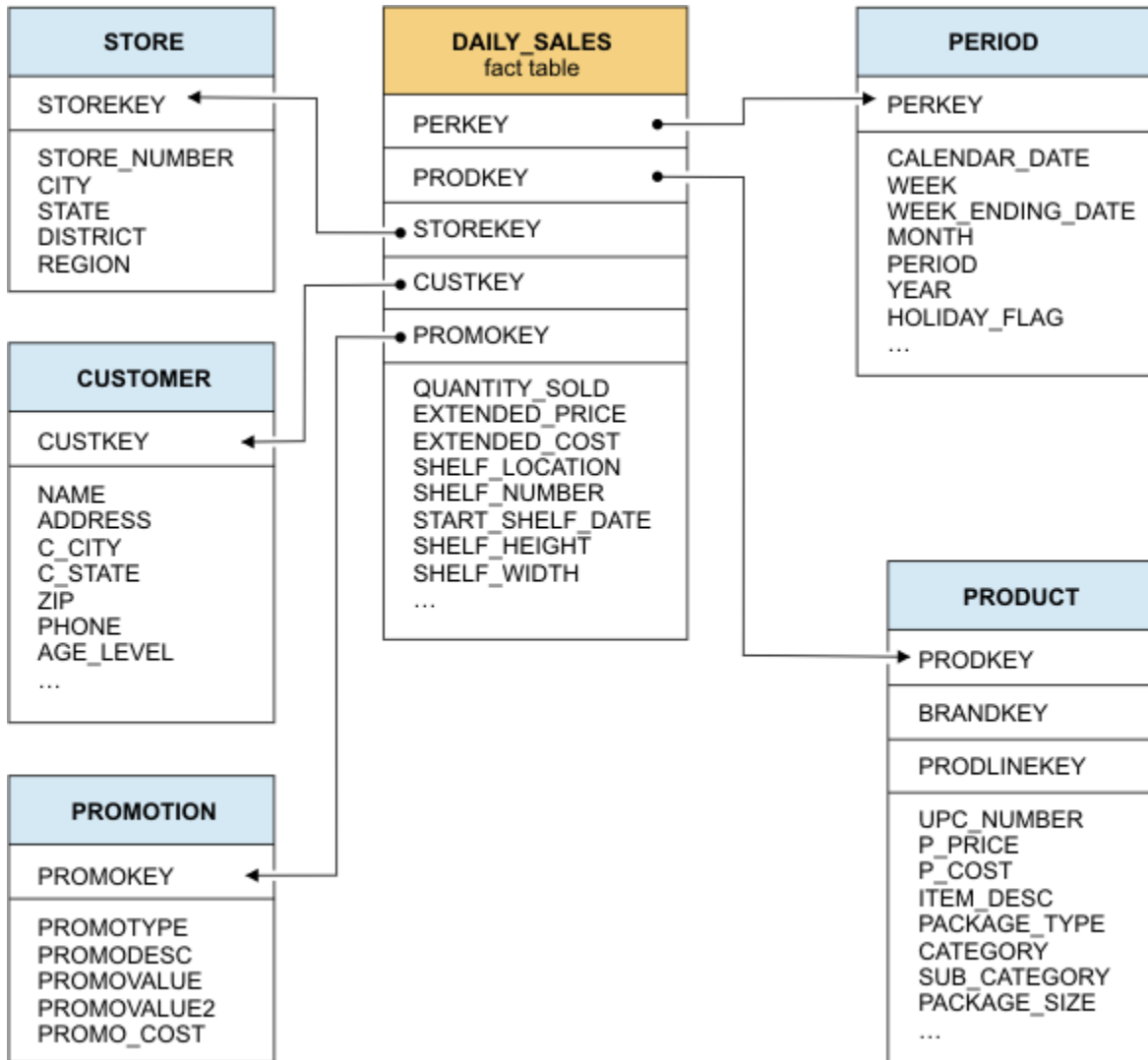
Figure 1. A sample snowflake schema which has the DAILY_SALES table as the fact table.



Data mart

A database that is oriented towards one or more specific subject areas of a business, such as tracking inventories or transactions, rather than an entire enterprise. A data mart is used by individual departments or groups. Like a data warehouse, you typically use a dimensional data model to build a data mart. For example the data mart might use a single star schema comprised of one fact table and several dimension tables. The design of a data mart often starts with an analysis of what data the user needs rather than focusing on the data that already exists.

Figure 2. A data mart with the DAILY_SALES fact table



Operational data store

A subject-oriented system that is optimized for looking up one or two records at a time for decision making. An operational data store (ODS) is a hybrid form of data warehouse that contains timely, current, integrated information. Including the ODS in the data warehousing environment enables access to more current data more quickly, particularly if the data warehouse is updated by one or more batch processes rather than updated continuously. The data typically is of a higher level granularity than the transaction. You can use an ODS for clerical, day-to-day decision making. This data can serve as the common source of data for data warehouses.

Why build a dimensional database?

In a data warehousing environment, the relational databases need to be optimized for data retrieval and tuned to support the analysis of business trends and projections.

This type of informational processing is known as online analytical processing (OLAP) or decision support system (DSS) processing. OLAP is also the term that database designers use to describe a dimensional approach to informational processing.

A dimensional database needs to be designed to support queries that retrieve a large number of records and that summarize data in different ways. A dimensional database tends to be subject oriented and aims to answer questions such as, What products are selling well? At what time of year do certain products sell best? In what regions are sales weakest?

In a dimensional data model, the data is represented as either facts or dimensions. A *fact* is typically numeric piece of data about a transaction, such as the number of items ordered. A *dimension* is the reference information about the numeric facts, such as the name of the customer. Any new data that you load into the dimensional database is usually updated in a batch, often from multiple sources.

Relational databases optimized for online transaction processing (OLTP) are designed to meet the day-to-day operational needs of the business. OLTP systems tend to organize data around specific processes, such as order entry. The database performance is tuned for those operational needs by using a *normalized data model* which stores data by using database normalization rules. Consequently, the database can retrieve a small number of records very quickly.

Some of the advantages of the dimensional data model are that data retrieval tends to be very quick and the organization of the data warehouse is easier for users to understand and use.

If you attempt to use a database that is designed for OLTP as your data warehouse, query performance will be very slow and it will be difficult to perform analysis on the data.

The following table summarizes the key differences between OLTP and OLAP databases:

Normalized database (OLTP)	Dimensional database (OLAP)
Data is atomized	Data is summarized
Data is current	Data is historical
Processes one record at a time	Processes many records at a time
Process oriented	Subject oriented
Designed for highly structured repetitive processing	Designed for highly unstructured analytical processing

Many of the problems that businesses attempt to solve are multidimensional in nature. For example, SQL queries that create summaries of product sales by region, region sales by product, and so on, might require hours of processing on an OLTP database. However, a dimensional database could process the same queries in a fraction of the time.

Besides the characteristic schema design differences between OLTP and OLAP databases, the query optimizer typically should be tuned differently for these two types of tasks. For example, in OLTP operations, the OPTCOMPIND setting (as specified by the environment variable or by the configuration parameter of that name) should typically be set to zero, to favor nested-loop joins. OLAP operations, in contrast, tend to be more efficient with an OPTCOMPIND setting of 2 to favor hash-

join query plans. For more information, see the **OPTCOMPIND** environment variable and the OPTCOMPIND configuration parameter. See the *Informix® Performance Guide* for additional information about OPTCOMPIND, join methods, and the query optimizer.

also supports the SET ENVIRONMENT OPTCOMPIND statement to change OPTCOMPIND setting dynamically during sessions in which both OLTP and OLAP operations are required. See the *Informix® Guide to SQL: Syntax* for more information about the SET ENVIRONMENT statement of SQL.

Informix® is designed to help businesses better leverage their existing information assets as they move into an on-demand business environment. In this type of environment, mission-critical database management applications typically require combination systems. The applications need both online transaction processing (OLTP), and batch and decision support systems (DSS), including online analytical processing (OLAP).

What is dimensional data?

Traditional relational databases, such as OLTP databases, are organized around a list of records. Each record contains related information that is organized into attributes (fields). The **customer** table of the **stores_demo** demonstration database, which includes fields for name, company, address, phone, and so forth, is a typical example. While this table has several fields of information, each row in the table pertains to only one customer. If you wanted to create a two-dimensional matrix with customer name and any other field, for example, phone number), you would realize that there is only a one-to-one correspondence. The following table is an example of a database table with fields that have only a one-to-one correspondence.

Table 1. A table with a one-to-one correspondences between fields

Customer	Phone number --->
Ludwig Pauli	408-789-8075
Carole Sadler	415-822-1289
Philip Currie	414-328-4543

You could put any combination of fields from the preceding **customer** table in this matrix, but you would always end up with a one-to-one correspondence, which shows that this table is not multidimensional and would not be well suited for a dimensional database.

However, consider a relational table that contains more than a one-to-one correspondence between the fields of the table. Suppose you create a table that contains sales data for products sold in each region of the country. For simplicity, the company has three products that are sold in three regions. The following table shows how you might store this data in a table, using a normalized data model. This table lends itself to multidimensional representation because it has more than one product per region and more than one region per product.

Table 2. A simple table with a many-to-many correspondence

Product	Region	Unit Sales
Football	East	2300
Football	West	4000
Football	Central	5600
Tennis racket	East	5500
Tennis racket	West	8000
Tennis racket	Central	2300
Baseball	East	10000
Baseball	West	22000
Baseball	Central	34000

Although this data can be forced into the three-field relational table, the data fits more naturally into the two-dimensional matrix in the following table. This matrix better represents the many-to-many relationship of product and region data shown in the previous table.

Table 3. A simple two-dimensional example

Region		Central	East	West
Product	Football	5600	2300	4000
	Tennis Racket	2300	5500	8000
	Baseball	34000	10000	22000

The performance advantages of the dimensional model over the normalized model can be great. A dimensional approach simplifies access to the data that you want to summarize or compare. For example, using the dimensional model to query the number of products sold in the West, the database server finds the **West** column and calculates the total for all row values in that column. To perform the same query on the normalized table, the database server has to search and retrieve each row where the **Region** column equals 'West' and then aggregate the data. In queries of this kind, the dimensional table can total all values of the **West** column in a fraction of the time it takes the relational table to find all the 'West' records.

Design a dimensional data model

To build a dimensional database, you start by designing a dimensional data model for your business.

You will learn how a dimensional model differs from a transactional model, what fact tables and dimension tables are and how to design them effectively. You will learn how to analyze the business processes in your organization where data is gathered and use that analysis to design a model for your dimensional data.

HCL Informix® includes several demonstration databases that are the basis for many examples in Informix® publications, including examples in the *Informix® Data Warehouse Guide*. The **stores_demo** database illustrates a relational schema with information about a fictitious wholesale sporting-goods distributor. You will use SQL and the data in the **stores_demo** database to populate a new dimensional database. The dimensional database is based on the simple dimensional data model that you learned about.

To understand the concepts of dimensional data modeling, you should have a basic understanding of SQL and relational database theory. This section provides only a summary of data warehousing concepts and describes a simple dimensional data model.

Related reference

[Implement a dimensional database on page 34](#)

Related information

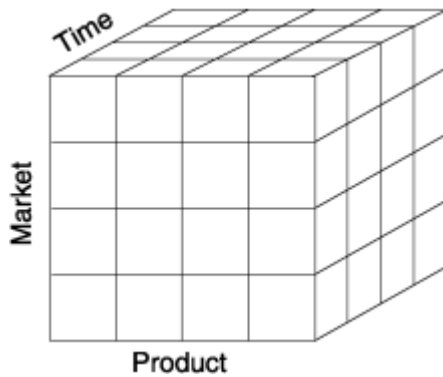
[Performance tuning dimensional databases on page 44](#)

Concepts of dimensional data modeling

To build a dimensional database, you start with a dimensional data model. The dimensional data model provides a method for making databases simple and understandable. You can conceive of a dimensional database as a database *cube* of three or four dimensions where users can access a slice of the database along any of its dimensions. To create a dimensional database, you need a model that lets you visualize the data.

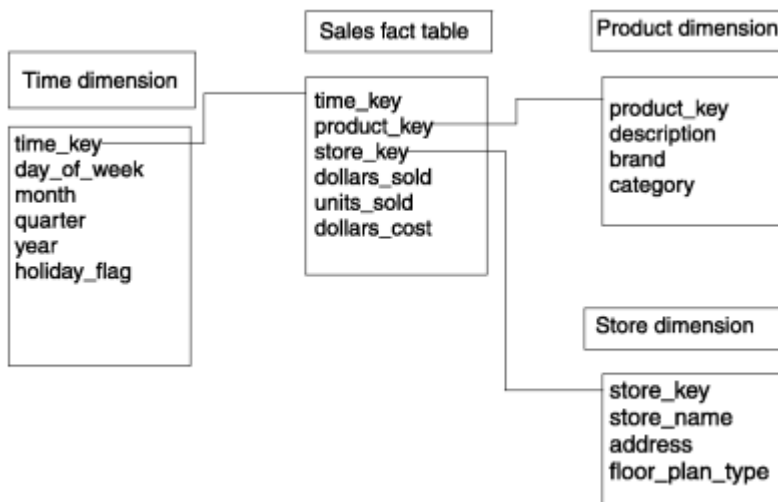
Suppose your business sells products in different markets and you want to evaluate the performance over time. It is easy to conceive of this business process as a cube of data, which contains dimensions for time, products, and markets. The following figure shows this dimensional model. The various intersections along the lines of the cube would contain the *measures* of the business. The measures correspond to a particular combination: product, market, and time data.

Figure 3. A dimensional model of a business that has time, product, and market dimensions



Another name for the dimensional model is the *star schema*. The database designers use this name because the diagram for this model looks like a star with one central table around which a set of other tables are displayed. The central table is the only table in the schema with multiple joins connecting it to all the other tables. This central table is called the *fact table* and the other tables are called *dimension tables*. The dimension tables all have only a single join that attaches them to the fact table, regardless of the query. The following figure shows a simple dimensional model of a business that sells products in different markets and evaluates business performance over time.

Figure 4. A typical dimensional model



The fact table

The fact table stores the measures of the business and points to the key value at the lowest level of each dimension table. The *measures* are quantitative or factual data about the subject.

The measures are generally numeric and correspond to the "how much" or "how many" aspects of a question. Examples of measures are price, product sales, product inventory, revenue, and so forth. A measure can be based on a column in a table or it can be calculated.

The following table shows a fact table whose measures are sums of the units sold, the revenue, and the profit for the sales of that product to that account on that day.

Table 4. A fact table with sample records

Product Code	Account code	Day code	Units sold	Revenue	Profit
1	5	32104	1	82.12	27.12
3	17	33111	2	171.12	66.00
1	13	32567	1	82.12	27.12

Before you design a fact table, you must determine the *granularity* of the fact table. The granularity corresponds to how you define an individual low-level record in that fact table. The granularity might be the individual transaction, a daily snapshot, or a monthly snapshot. The fact table shown contains one row for every product sold to each account each day. Thus, the granularity of the fact table is expressed as *product by account by day*.

Dimensions of the data model

A *dimension* represents a single set of objects or events in the real world. Each dimension that you identify for the data model gets implemented as a dimension table. Dimensions are the qualifiers that make the measures of the fact table meaningful, because they answer the what, when, and where aspects of a question. For example, consider the following business questions, for which the dimensions are italicized:

- What *accounts* produced the highest revenue last *year*?
- What was our profit by *vendor*?
- How many units were sold for each *product*?

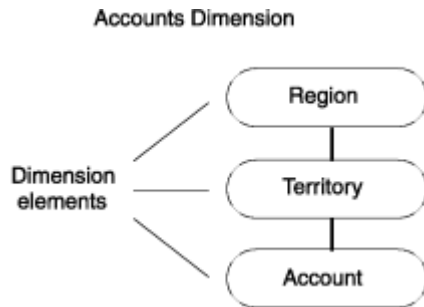
In the preceding set of questions, revenue, profit, and units sold are measures (not dimensions), as each represents quantitative or factual data.

Dimension elements

A dimension can define multiple *dimension elements* for different levels of summation.

For example, all the elements that relate to the structure of a sales organization might comprise one dimension. The following figure shows the dimension elements that the **Accounts** dimension defines.

Figure 5. Dimension elements in the accounts dimension



Dimensions are made up of hierarchies of related elements. Because of the hierarchical aspect of dimensions, users are able to construct queries that access data at a higher level (*roll up*) or lower level (*drill down*) than the previous level of detail. The figure shows the hierarchical relationships of the dimension elements:

- The account elements roll up to the territory elements
- The territory elements roll up to the region elements

Users can query at different levels of the dimension, depending on the data they want to retrieve. For example, users might perform a query against all regions and then drill down to the territory or account level for detailed information.

Dimension elements are usually stored in the database as numeric codes or short character strings to facilitate joins to other tables.

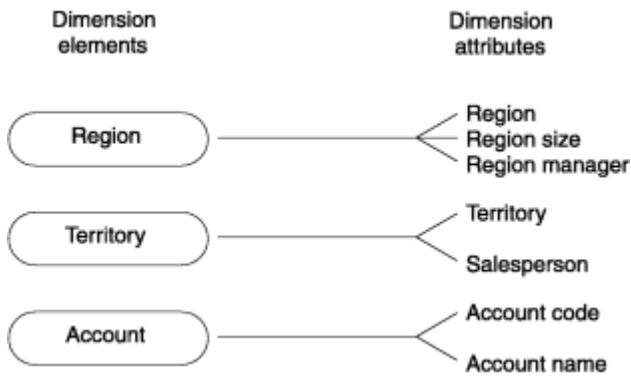
Each dimension element can define multiple dimension attributes, in the same way dimensions can define multiple dimension elements.

Dimension attributes

A *dimension attribute* is a column in a dimension table. Each attribute describes a level of summary within a dimension hierarchy.

The dimension elements define the hierarchical relationships within a dimension table. The dimension attributes describe the dimension elements in terms that are familiar to users. The following figure shows the dimension elements and corresponding attributes of the **Account** dimension.

Figure 6. Attributes that correspond to the dimension elements



Because dimension attributes describe the items in a dimension, they are most useful when they are text.

i Tip: Sometimes during the design process, it is unclear whether a numeric data field from a production data source is a measured fact or an attribute. Generally, if the numeric data field is a measurement that changes each time you sample it, the field is a fact. If field is a discretely valued description of something that is more or less constant, it is a dimension attribute.

Dimension tables

A *dimension table* is a table that stores the textual descriptions of the dimensions of the business. A dimension table contains an element and an attribute, if appropriate, for each level in the hierarchy.

The lowest level of detail that is required for data analysis determines the lowest level in the hierarchy. Levels higher than this base level store redundant data. This denormalized table reduces the number of joins that are required for a query and makes it easier for users to query at higher levels and then drill down to lower levels of detail. The term *drilling down* means to add row headers from the dimension tables to your query. The following table shows an example of a dimension table that is based on the **Account** dimension.

Table 5. An example of a dimension table

Acct code	Account name	Territory	Salesman	Region	Region size	Region manager
1	Javier's Mfg.	101	B. Gupta	Asia-Pacific	Over 50	T. Sent
2	TBD Sales	101	B. Gupta	Asia-Pacific	Over 50	T. Sent
3	Tariq's Wares	101	B. Gupta	Asia-Pacific	Over 50	T. Sent
4	The Golf Co.	201	S. Chiba	Asia-Pacific	Over 50	T. Sent

Building a dimensional data model

About this task

To build a dimensional data model, you need a methodology that outlines the decisions you need to make to complete the database design. This methodology uses a top-down approach because it first identifies the major processes in your organization where data is collected. An important task of the database designer is to start with the existing sources of data that your organization uses. After the processes are identified, one or more fact tables are built from each business process. The following steps describe the methodology you use to build the data model.

A dimensional database can be based on multiple business processes and can contain many fact tables. However, to focus on the concepts, the data model that this section describes is based on a single business process and has one fact table.

To build a dimensional database:

1. Choose the business processes that you want to use to analyze the subject area to be modeled.
2. Determine the granularity of the fact tables.
3. Identify dimensions and hierarchies for each fact table.
4. Identify measures for the fact tables.
5. Determine the attributes for each dimension table.
6. Get users to verify the data model.

A business process

A *business process* is an important operation in your organization that some legacy system supports. You collect data from this system to use in your dimensional database.

The business process identifies what end users are doing with their data, where the data comes from, and how to transform that data to make it meaningful. The information can come from many sources, including finance, sales analysis, market analysis, customer profiles. The following list shows different business processes you might use to determine what data to include in your dimensional database:

- Sales
- Shipments
- Inventory
- Orders
- Invoices

Summary of a business process

Suppose your organization wants to analyze customer buying trends by product line and region so that you can develop more effective marketing strategies. In this scenario, the subject area for your data model is **sales**.

After many interviews and thorough analysis of your sales business process, your organization collects the following information:

- Customer-base information has changed.

Previously, sales districts were divided by city. Now the customer base corresponds to two regions: Region 1 for California and Region 2 for all other states.

- The following reports are most critical to marketing:

- Monthly revenue, cost, net profit by product line from each vendor
- Revenue and units sold by product, by region, and by month
- Monthly customer revenue
- Quarterly revenue from each vendor

- Most sales analysis is based on monthly results, but you can choose to analyze sales by week or accounting period (at a later date).

- A data-entry system exists in a relational database.

To develop a working data model, you can assume that the relational database of sales information has the following properties:

- The **stores_demo** database provides much of the revenue data that the marketing department uses.
 - The product code that analysts use is stored in the **catalog** table by the catalog number.
 - The product line code is stored in the **stock** table by the stock number. The product line name is stored as description.
 - The product hierarchies are somewhat complicated. Each product line has many products, and each manufacturer has many products.
- All the cost data for each product is stored in a flat file named **costs.lst** on a different purchasing system.
 - Customer data is stored in the **stores_demo** database.

The region information has not yet been added to the database.

An important characteristic of the dimensional model is that it uses business labels familiar to end users rather than internal tables or column names. After the analysis of the business process is completed, you should have all the information you need to create the measures, dimensions, and relationships for the dimensional data model. This dimensional data model is used to implement the **sales_demo** database that the section [Implement a dimensional database on page 34](#) describes.

The **stores_demo** demonstration database is the primary data source for the dimensional data model that this section builds. For detailed information about the data sources that are used to populate the tables of the **sales_demo** database, see [Mapping data from data sources to the database on page 36](#).

Determine the granularity of the fact table

After you gather all the relevant information about the subject area, the next step in the design process is to determine the granularity of the fact table.

To do this you must decide what an individual low-level record in the fact table should contain. The components that make up the granularity of the fact table correspond directly with the dimensions of the data model. Therefore, when you define the granularity of the fact table, you identify the dimensions of the data model.

How granularity affects the size of the database

The granularity of the fact table also determines how much storage space the database requires.

For example, consider the following possible granularities for a fact table:

- Product by day by region
- Product by month by region

The size of a database that has a granularity of product by day by region would be much greater than a database with a granularity of product by month by region. The database contains records for every transaction made each day as opposed to a monthly summation of the transactions. You must carefully determine the granularity of your fact table because too fine a granularity could result in an astronomically large database. Conversely, too coarse a granularity could mean the data is not detailed enough for users to perform meaningful queries against the database.

Use the business process to determine the granularity

A careful review of the information gathered from the business process should provide what you need to determine the granularity of the fact table. To summarize, your organization wants to analyze customer-buying trends by product line and region so that you can develop more effective marketing strategies.

Customer by product level granularity

The granularity of the fact table should always represent the lowest level for each corresponding dimension.

When you review the information from the business process, the granularity for customer and product dimensions of the fact table are apparent. Customer and product cannot be reasonably reduced any further. These dimensions already express the lowest level of an individual record for the fact table. In some cases, product might be further reduced to the level of product component because a product could be made up of multiple components.

Customer by product by district level granularity

Because the customer buying trends that your organization wants to analyze include a geographical component, you still need to decide the lowest level for the region information.

The business process indicates that in the past, sales districts were divided by city, but now your organization distinguishes between two regions for the customer base:

- Region 1 for California
- Region 2 for all other states

Nonetheless, at the lowest level, your organization still includes sales district data. The district represents the lowest level for geographical information and provides a third component to further define the granularity of the fact table.

Customer by product by district by day level granularity

Customer-buying trends always occur over time, so the granularity of the fact table must include a time component.

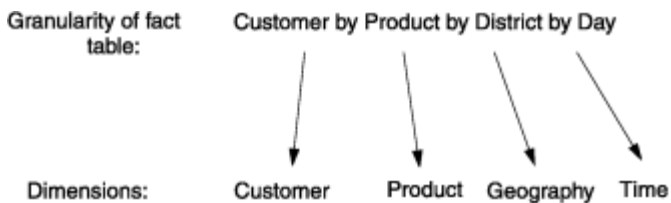
Suppose your organization decides to create reports by week, accounting period, month, quarter, or year. At the lowest level, you probably want to choose a base granularity of day. This granularity allows your business to compare sales on Tuesdays with sales on Fridays, compare sales for the first day of each month, and so forth. The granularity of the fact table is now complete.

The decision to choose a granularity of day means that each record in the **time** dimension table represents a day. In terms of the storage requirements, even 10 years of daily data is only about 3,650 records, which is a relatively small dimension table.

Identify the dimensions and hierarchies

After you determine the granularity of the fact table, it is easy to identify the primary dimensions for the data model because each component that defines the granularity corresponds to a dimension.

The following figure shows the relationship between the granularity of the fact table and the dimensions of the data model. Figure 7. The granularity of the fact table corresponds to the dimensions of the data model

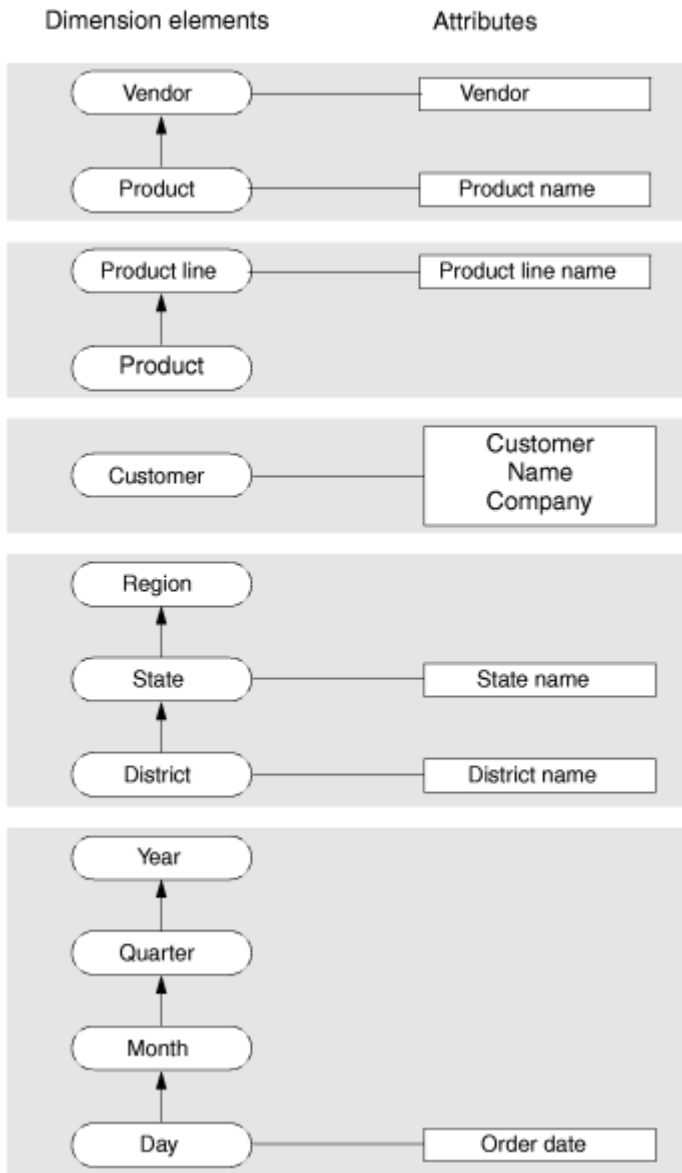


With the dimensions (customer, product, geography, time) for the data model in place, the schema diagram begins to take shape.

i Tip: At this point, you can add additional dimensions to the primary granularity of the fact table, where the new dimensions take on only a single value under each combination of the primary dimensions. If you see that an additional dimension violates the granularity because it causes additional records to be generated, then you must revise the granularity of the fact table to accommodate the additional dimension. For this data model, no additional dimensions need to be added.

You can now map out dimension elements and hierarchies for each dimension. The following figure shows the relationships among dimensions, dimension elements, and the inherent hierarchies.

Figure 8. The relationships between dimensions, dimension elements, and the inherent hierarchies



In most cases, the dimension elements need to express the lowest possible granularity for each dimension, not because queries need to access individual low-level records, but because queries need to cut through the database in precise ways. In other words, even though the questions that a data warehousing environment poses are usually broad, these questions still depend on the lowest level of product detail.

Product dimension

The dimension elements for the **product** dimension are product, product line, and vendor:

- Product has a roll-up hierarchical relationship with product line and with vendor. Product has an attribute of product name.
- Product line has an attribute of product line name.
- Vendor has an attribute of vendor.

Customer dimension

The dimension element for the customer dimension is customer, which has attributes of customer, name, and company.

Geography dimension

The dimension elements for the geography dimension are district, state, and region:

- District has a roll-up hierarchical relationship with state, which has a roll-up hierarchical relationship with region.
- District has an attribute of district name.
- State has an attribute of state name.

Time dimension

The dimensional elements for the time dimension are day, month, quarter, and year.

- Day has a roll-up hierarchical relationship with month, which has a roll-up hierarchical relationship with quarter, which has a roll-up hierarchical relationship with year.
- Day has an attribute of order date.

Establish referential relationships

For the database server to support the dimensional data model, you must define logical dependencies between the fact tables and their dimension tables.

These logical dependencies should be reflected in the columns and indexes that you include in the schema of each table, and in the referential constraints that you define between each fact table and the associated dimension tables. For the large fragmented tables in typical data warehousing operations, these logical dependencies can be the basis for:

- Fragment-key expressions
- Join conditions
- Query predicates for fragment elimination

These query components can significantly improve the performance and throughput of the data warehouse.

A referential constraint enforces a one-to-one relationship between the values in referencing columns (of the foreign key) and the referenced columns (of the primary key or unique constraint). The relationship between the referenced table with the primary key constraint and the referencing table with the foreign key constraint is sometimes called a *parent-child relationship*. The corresponding columns of the parent and child tables can have the same identifiers, but having the same identifiers is not a requirement. There can also be a many-to-one relationship between the referencing table (with the foreign key) and the referenced table (with the primary key, or with the unique constraint).

In the dimensional model, a primary key constraint or a unique constraint in the fact table corresponds to a foreign key constraint in the dimension table. These constraints are specified in the CREATE TABLE or ALTER TABLE statements of SQL that defines the schema of the tables. Because the tables in the primary key and foreign key constraints must be in the same database, the database schema must include the dimension tables of each fact table.

The same data values can appear in the constrained columns of both tables. As a result, the index on which these referential constraints are defined can be used in queries as join predicates to join the fact table and the dimensional table.

For tables that are fragmented by expression or fragmented by list, you can use the foreign key as the fragmentation key for the dimension tables. If you use the foreign key as the fragmentation key, you can use the equality operator or MATCHES operator with the primary key and foreign key values as the join predicate in queries and other data manipulation operations. The join predicate will be `TRUE` for only a subset of the fact table fragments. As a result, the query optimizer can use fragment elimination to process only the fact table partitions that contain qualifying rows.

Resisting normalization

Efforts to normalize a dimensional database can actually prohibit an efficient dimensional design.

If the four foreign keys of the fact table are tightly administered consecutive integers, you could reserve as little as 16 bytes for all four keys (4 bytes each for time, product, customer, and geography) of the fact table. If the four measures in the fact table were each 4-byte integer columns, you would need to reserve only another 16 bytes. Thus, each record of the fact table would be only 32 bytes. Even a billion-row fact table would require only about 32 gigabytes of primary data space.

With its compact keys and data, such a storage-lean fact table is typical for dimensional databases. The fact table in a dimensional model is by nature highly normalized. You cannot further normalize the extremely complex many-to-many relationships among the four keys in the fact table because no correlation exists between the four dimension tables. Virtually every product is sold every day to all customers in every region.

The fact table is the largest table in a dimensional database. Because the dimension tables are usually much smaller than the fact table, you can ignore the dimension tables when you calculate the disk space for your database. Efforts to normalize any of the tables in a dimensional database solely to save disk space are pointless. Furthermore, normalized dimension tables undermine the ability of users to explore a single dimension table to set constraints and choose useful row headers.

Choose the attributes for the dimension tables

After you complete the fact table, you can decide the dimension attributes for each of the dimension tables. To illustrate how to choose the attributes, consider the **time** dimension. The data model for the sales business process defines a granularity of day that corresponds to the time dimension, so that each record in the **time** dimension table represents a day. Keep in mind that each field of the table is defined by the particular day the record represents.

The analysis of the sales business process also indicates that the marketing department needs monthly, quarterly, and annual reports, so the time dimension includes the elements: day, month, quarter, and year. Each element is assigned an attribute that describes the element and a code attribute, to avoid column values that contain long character strings. The following table shows the attributes for the **time** dimension table and sample values for each field of the table.

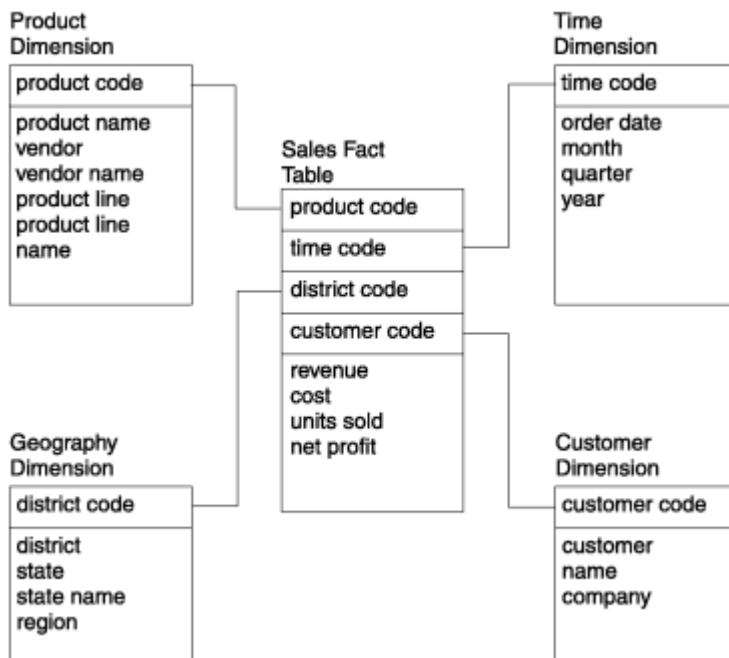
Table 6. Attributes for the time dimension

time code	order date	month code	month	quarter code	quarter	year
35276	07/31/2010	7	july	3	third q	2010
35277	08/01/2010	8	aug	3	third q	2010
35278	08/02/2010	8	aug	3	third q	2010

The previous table shows that the attribute names you assign should be familiar business terms that make it easy for end users to form queries on the database.

The following figure shows the completed data model for the sales business process with all the attributes defined for each dimension table. The elements of the Sales fact table are: product code, time code, district code, customer code, revenue, cost, units sold, and net profit. Some of these elements join the Sales fact table to the dimension tables. Additional elements for each dimension table have been identified.

Figure 9. The completed dimensional data model for the sales business process



Product dimension table

The product code element joins the Sales fact table to the Product dimension table. The additional elements in the Product dimension table are: product name, vendor, vendor name, product line, and product line name.

Time dimension table

The time code element joins the Sales fact table to the Time dimension table. The additional elements in the Time dimension table are: order date, month, quarter, and year.

Geography dimension table

The district code element joins the Sales fact table to the Geography dimension table. The additional elements in the Geography dimension table are: district, state, state name, and region.

Customer dimension table

The customer code element joins Sales fact table to Customer dimension table. The additional elements in the Customer dimension table are: customer name and company.

Fragmentation: Storage distribution strategies

The performance of data warehousing applications can typically benefit from distributed storage allocation designs for partitioning a database table into two or more fragments. Each fragment has the same schema as the table, and stores a subset of the rows in the table (rather than a subset of its columns).

The fragments of a table can be stored in dbspaces on different devices, or in dbspaces on the same physical storage device. The fragments can also be stored in named partitions within a single dbspace.

A database can include both fragmented and nonfragmented tables. Index storage can also be fragmented, either in the same storage spaces as their table (called *attached indexes*) or in a different storage distribution scheme (*detached indexes*).

Potential performance and security advantages of distributed storage include these:

- For frequently-accessed tables, fragmentation can reduce the overhead of I/O contention for data that resides on a single storage device.
- The GRANT FRAGMENT and REVOKE FRAGMENT statements of SQL can specify the access privileges that users, roles, or the PUBLIC group hold on specified fragments of the table. With appropriate fragmentation strategies, these statements can selectively restrict user access to subsets of the records in a table.
- For databases that enables parallel-database queries (PDQ), multiple scan threads require less time to scan the fragments than to scan the same rows in a nonfragmented table.
- Input operations that distribute new rows across multiple fragments run more quickly (using multiple INSERT threads) than if a single table extent stores the same rows.
- For fragmentation strategies where the storage allocation of rows is correlated with data values, query execution plans can ignore fragments that are logically excluded by predicates in the query. Defining fragments to improve selectivity is called *fragment elimination*.
- In cluster environments, fragmentation can reduce the time required for recovery from hardware failure, because restoring only a subset of the fragments imposes a smaller data load than restoring the entire table.
- For tables fragmented by interval, the database server create new fragments automatically, simplifying management of the data.



Note: Do not confuse table fragmentation strategies, which can improve the efficiency and throughput of database operations, with the various pejorative meanings of *fragmentation* in reference to file systems that waste storage



space or increase retrieval time through inefficient storage algorithms, or through insufficient use of defragmentation tools to store files in contiguous disk partitions.

Informix® fragmentation options

Informix® supports the following storage fragmentation strategies that can be applied to database tables:

By Round-robin

A specified number of fragments is defined for the table. Inserted rows are automatically distributed for storage in these fragments, without regard to data values in the row, in order to balance the number of rows in each fragment. Such fragments are called *round-robin fragments*.

By Expression

Each fragment is defined by a Boolean expression that can be evaluated for one or more columns of the table. Inserted rows are stored in a fragments for which the expression that defines the fragment is true for the data in that row. Rows that match the expression for more than one fragment are stored in the first matching fragment within the ordered list of fragments that the system catalog maintains for the table. Such fragments are called *expression fragments*.

By List

Each fragment is defined by a list of one or more constant values that correspond to one or more columns in the table. No two fragments can share the same value in their lists. These values must be categories on a nominal scale that has no quantified order within the set of categories. Inserted rows are stored in the fragment that matches the data value of one or more columns. Such fragments are called *list fragments*.

By Interval

At least one fragment must be defined for values less than a numeric, DATE, or DATETIME column in the table. An interval size, specifying the range of fragment key values assigned to a single fragment, must also be defined. You can optionally specify a list of dbspaces to store interval fragments. The fragments created by the user when the fragmentation strategy is defined are called *range fragments*. The database server automatically creates new fragments of the same interval size to store rows whose fragment key values are outside the range of the user-defined range fragments. Fragments created by the database server are called *interval fragments*.

Each user-defined permanent or temporary database table can either be *nonfragmented* or else can have exactly one fragmentation scheme. You cannot, for example, define a table in which some fragments use a round-robin strategy, and other fragments use a list or interval strategy.

You can use the ALTER FRAGMENT statement of SQL, however, to modify the fragmentation scheme of a table in various ways, including these:

- to change the fragmentation strategy of a fragmented table,
- to define a fragmentation strategy for a nonfragmented table,
- to change a fragmented table to a nonfragmented table,
- to add another fragment to an existing fragmented table,
- to combine two tables that have identical structures into a single fragmented table,

- to drop one or more dbspaces from the list of dbspaces that store interval fragments.
- to detach one fragment from a fragmented table and store the rows in a new nonfragmented table.

For more information about the ALTER FRAGMENT statement and some of the tasks that it can accomplish in data warehousing operations, see the [Change the storage distribution strategy on page 40](#).

Storage fragmentation terms

The following terms are useful for understanding and using the various strategies available for the distributed storage of table and index fragments.

Fragment key

The column or a set of columns on which the table or index is fragmented. Depending on the chosen fragmentation strategy, the fragment key can be a column, or a single column expression, or a multi-column expression. For a row inserted into a table for which a fragment key is defined, the value of the column (or the set for values in the fragment key columns) determines which fragment stores the row. A synonym for fragment key is *partitioning key*. Tables partitioned by round-robin have no fragment key.

Fragment list

An ordered list of the fragments that the database server maintains for every fragmented table or index. By default, the ordinal positions of each fragment on this list reflects the sequence in which the fragments were created. The system catalog stores this integer value in the **sysfragments.evalpos** column of the row that describes the fragment. Queries that do not use fragment elimination read the fragments in ascending order of their **evalpos** values. The database server automatically updates **evalpos** values to reflect changes to the fragment list. Updates to the list are required, for example, when the database server creates an interval fragment, or when the ALTER FRAGMENT statement of SQL adds new fragments, or drops or modifies existing fragments.

Fragment expression

An expression that defines a specific fragment. For example, if the fragment key is **colA** of data type SMALLINT, a fragment could be defined by the expression `colA <=8 OR colA IN (9,10,21,22,23)` in an expression based fragmentation strategy.

- Expression-based fragments are defined by a Boolean expression.
- List-based fragments are defined by one or more constant expressions.
- Range fragments (in interval fragmentation) are defined by a range expression. The only valid operator in the range expression is the less-than (`<`) operator. (For example, `VALUES < 100`).
- System-defined interval fragments (in interval fragmentation) are defined by a system-generated expression that includes the greater-than-or-equal `>=` relational operator, the `AND` Boolean operator, and the less-than (`<`) relational operator. (For example, `VALUES >= 100 AND VALUES < 300` specifies an interval that includes fragmentation key values ranging from 100 to the (non-inclusive) upper limit of 300.)

Tables partitioned by round-robin have no fragment expressions.

NULL fragment

A fragment that stores NULL values (either because its range fragment or list fragment expression is `IS NULL`, or because a list-based or expression-based fragment is defined with NULL as its fragment expression). For all fragmentation strategies except round-robin, the database server returns an exception if you insert a row whose fragment key value is missing, but no NULL fragment is defined (and for list or expression strategies, no REMAINDER fragment is defined). You do not need to define a NULL fragment if the fragment key column enforces a NOT NULL constraint.

REMAINDER fragment

A fragment that stores any row whose fragment key value does not match the fragment expression of any fragment. If you attempt to insert a row that does not match any fragment key value for a table or index that is fragmented by expression or by list, and no REMAINDER fragment is defined, the database server issues an exception. You cannot define a REMAINDER fragment for tables fragmented by a round-robin or interval strategy.

Transition fragment

In an interval fragmentation scheme, the range fragment whose upper limit in its VALUES clause is larger than the upper limit for any other range fragment. If no interval fragments have been created for the table, inserting a row whose fragment-key value exceeds that upper limit requires the database server to create a new interval fragment. The upper limit of the transition fragment VALUES clause is called the *transition value* for the table.

The MODIFY INTERVAL TRANSITION option to the ALTER FRAGMENT statement can increase the transition value for a table. This can result in a different fragment becoming the new transition fragment. This and other ALTER FRAGMENT operations can cause changes to column values in the **sysfragments** system catalog table for the transition fragment, including these:

- its position relative to other fragments (the **evalpos** column),
- its fragment expression (the **exprtext** and **exprbin** columns),
- and its name (the **partition** column).

Fragmentation by ROUND ROBIN

For a table that uses a round-robin distribution scheme, the rows that the database server stores in an insert or load operation are distributed cyclically among a user-defined number of fragments, so that the number of rows inserted into each fragment is approximately the same (± 100).

Round-robin distributions are also called *even* distributions, because the design goal of this strategy is for an evenly balanced distribution among the fragments.

The syntax for defining round-robin interval fragmentation requires that you specify at least two round-robin fragments in one of two forms. This form defines round-robin fragments and declare a name for each fragment:

```
FRAGMENT BY ROUND ROBIN
PARTITION partition IN dbspace,
```

```

. . .
PARTITION partition IN dbspace

```

As in other fragmentation schemes, each `PARTITION partition` specification declares the name of a fragment, which must be unique among the names of fragments of the same table. The `dbspace` specification can be different for each fragment, or some fragments (or all of the fragments) can be stored in separate named partitions of the same `dbspace`. Each `partition` is the name of a round-robin fragment.

This alternative form defines round-robin fragments with no explicit name:

```

FRAGMENT BY ROUND ROBIN IN dbspace_list

```

Here the `dbspace_list` specification is a comma-separated list of at least 2 (but no more than 2048) `dbspaces`, each of which stores a single round-robin fragment. No `dbspace` can appear more than once in this list. (In the system catalog, the `sysfragments.partition` column stores the identifier of the fragment. For fragments defined without the `PARTITION` keyword, the `partition` value is the identifier of the `dbspace` where the fragment is stored. For this reason, a repeated `dbspace` in `dbspace_list` violates a uniqueness requirement for names of fragments of the same table.)

A round-robin distribution scheme must be defined by only one or the other of these two syntax forms.

A table that is fragmented by round-robin has no fragment key, no fragment expressions, and no `REMAINDER` fragment. (An alternative description is that every round-robin fragment resembles a remainder fragment, because no fragment expressions are defined to match a fragment key for the inserted rows. But the `REMAINDER` keyword is not valid in the SQL syntax to define a round-robin distribution strategy.)

Because no fragment expressions are evaluated when the database server loads new rows into round-robin fragments, this strategy provides the best performance for insert operations.

Only tables, not indexes, can be defined with round-robin fragmentation. For performance reasons, any indexes that you define on a table that is fragmented by round-robin should be nonfragmented indexes.

Because a round-robin distribution strategy has no fragment key and no fragment expressions, you cannot explicitly define a `NULL` round-robin fragment. When rows with missing data are loaded into a table by round-robin, the rows with `NULL` values are stored wherever the database server happens to insert them as it approximately equalizes the number of inserted rows for every fragment.

By design, the `GRANT FRAGMENT` and `REVOKE FRAGMENT` statements of SQL cannot reference round-robin fragments. Because each fragment stores a quasi-random subset of the rows, the DBA cannot predict which rows will be stored in a given round-robin fragment. If some rows contain unencrypted sensitive information, table-level (rather than fragment-level) is a more appropriate granularity for granting or withholding discretionary access privileges in databases that do not implement label-based (LBAC) security policies.

Because round-robin fragments are uncorrelated with data values, queries of tables that are fragmented by round-robin cannot benefit from fragment elimination. Round-robin distribution schemes are useful for balancing the rows in a set of table fragments across multiple devices, but other storage distribution schemes are typically used in data warehouse applications that query dimensional tables, because the performance advantages of round-robin in loading data are more than offset by slower data retrieval from round-robin fragments.

Fragmentation by EXPRESSION

For a table that uses an expression-based distribution scheme, the rows that the database server stores in an insert or load operation are distributed among a user-defined number of fragments, in which each fragment is defined by a Boolean expression for the fragment key.

The fragment expression must be a column expression. This can be the same column (or the same set of columns) for all of the fragments, or different fragments can be defined with different keys. The expression can only reference columns in the table that is being fragmented. Subqueries or calls to user-defined routines are not valid.

The syntax for defining an expression fragmentation strategy defines one or more expression fragments of this form:

```
FRAGMENT BY EXPRESSION
  PARTITION partition expression IN dbspace,
  . . .
  PARTITION partition expression IN dbspace,
  PARTITION partition VALUES (NULL) IN dbspace,
  PARTITION partition REMAINDER IN dbspace
```

As in other fragmentation schemes, each `PARTITION partition` specification declares the unique name of a fragment. The `expression` specification defines the fragment expression, and the `IN dbspace` specification defines the storage location for the fragment. You can optionally define a NULL fragment by specifying `NULL` as the `expression`.

You also can optionally define a REMAINDER fragment for rows that match none of the specified fragment expressions. For some queries, the REMAINDER fragment might be difficult to eliminate, and for some tables, the REMAINDER fragment might become quite large, but the database server issues an exception if the fragment key value for an inserted row matches no fragment expression, and no REMAINDER fragment is defined.

You can optionally define a NULL fragment to stores rows in which the fragment key value is missing.

During an insert into a table that is fragmented by expression, the database server takes these actions:

1. The fragment key value for the row is evaluated.
2. The fragment expression for each fragment is evaluated and compared to the fragment key value for the row, beginning with the fragment whose `sysfragments.evalpos` value in the system catalog is lowest.
3. If there is no match, the previous step is repeated for the fragment with next highest `sysfragments.evalpos` value.
4. This continues until the first match is found between the fragment key value and a fragment expression, after which the row is stored in the matching fragment.
5. If no match is found in the entire list of fragments, the row is stored in the REMAINDER fragment. (In this case of a row with an unmatched fragment key, if no REMAINDER is defined, an exception is issued.)

For expression-based fragmentation schemes that define overlapping fragment expressions, the storage location of rows that match the fragment expression of more than one fragment is dependent on the `evalpos` value for that fragment. You can avoid this dependency by only defining non-overlapping fragment expressions.

The `evalpos` value of a fragment is determined by its position in the initial fragment list within the `FRAGMENT BY EXPRESSION` or `PARTITION BY EXPRESSION` clause that defined the storage distribution of the table. Any new fragments added by `ALTER FRAGMENT` operations are assigned, by default, the next higher `evalpos` value (and will therefore be evaluated last during `INSERT` operations) unless you explicitly specify a position with the `BEFORE` or `AFTER` keyword. In this case, the `evalpos` value for the new fragment will be the ordinal position where was inserted into the fragment list. For tables

that are fragmented by expression into a large number of fragments, you can achieve greater efficiency in INSERT and LOAD operations when fragments that are more likely to match fragment key values have relatively low **evalpos** values within the fragment list.

Fragmentation by expressions that creates nonoverlapping fragments on a single column can be an effective strategy for supporting fragment elimination in queries. The database server can eliminate fragments, for example, for queries with range expressions as well as queries with equality expressions if the query predicates correspond to fragment expressions. Expressions with relational operators and logical operators (or with both) can similarly be used for fragment expressions that match query filters.

Fragmentation by LIST

A list fragmentation strategy partitions data into a set of fragments that are each defined by a list of discrete values of the fragment key. Every expression must be a quoted string or a literal value. Each value in the list must be unique among the lists for fragments of the same object.

Fragmenting by list resembles fragmentation by expression (where the fragment expressions include the IN operator or the logical OR operator) in these respects:

- Every non-REMAINDER fragment stores rows for which the fragment key values matches the fragment expression.
- You can optionally define a REMAINDER fragment.
- You can optionally define a NULL fragment.

As the name implies, however, fragmentation by list defines each fragment by a list of fragment expressions, rather than restricting each fragment to a single expression.

The syntax for defining a list fragmentation strategy requires one or more list fragments of the following form.

```
FRAGMENT BY LIST
PARTITION partition VALUES (expression_list) IN dbspace,
. . .
PARTITION partition VALUES (expression_list) IN dbspace,
PARTITION partition VALUES (NULL) IN dbspace,
PARTITION partition REMAINDER IN dbspace
```

Here the last two partitions (whose expressions define a NULL fragment and a REMAINDER fragment) are not required.

As with other fragmentation schemes, each `PARTITION partition` specification declares the unique name of a fragment. The `(expression_list)` specification is the comma-separated list of one or more constant expressions that defines each list fragment, and the `IN dbspace` specification identifies the storage location of the fragment.

You can optionally define a NULL fragment by specifying `NULL` as the only expression in the `expression_list`. You cannot include `NULL` in an expression list with other values that define the same fragment.

An alternative syntax notation for defining the NULL fragment is `VALUES IS NULL` (with no delimiting parentheses) as the only expression for a fragment. The digit `0` is not equivalent to the NULL or IS NULL keywords.

Just as in expression-based fragmentation, you can optionally define a REMAINDER fragment for rows that match none of the specified fragment expressions. If you define a REMAINDER fragment but no NULL fragment, rows with the fragment

key value missing are stored in the REMAINDER fragment. The database server issues an exception for INSERT operations if the fragment key value for an inserted row matches no fragment expression, and no REMAINDER fragment is defined. An exception is similarly issued if data is missing from the fragment key column, but the fragment list includes no NULL fragment and no REMAINDER fragment.

When you use the CREATE INDEX statement to define an index on a table that is fragmented by list, it is not necessary to include the FRAGMENT BY or PARTITION BY clause to create indexes that use the same list fragmentation strategy as their table. By default, the database server partitions the index by the same list fragmentation strategy as its table, and declares for each index fragment the same name that you specified after the PARTITION keyword for the corresponding table fragment.

The most important difference between fragmentation by list and fragmentation by expression is that every value in the list for each fragment must be unique among all the expression lists that define fragments of the same table or index. The database server issues an error if the lists of expressions for two list fragments include the same fragment key value. This uniqueness requirement for fragment expressions simplifies fragment elimination in queries, if the fragment expressions correspond to query predicates and filters that support fragment elimination.

A list fragmentation strategy is most effective when the fragment key for a table has finite set of values, and queries on the table specify equality predicates on the fragment key. For a table whose fragment key is a numeric or time data type with a range of possible values that resembles a continuum, an interval fragmentation scheme is recommended, rather than list fragmentation.

Fragmentation by INTERVAL

An interval fragmentation strategy partitions data into fragments based on an interval value of the fragment key. The interval value must be a column expression that references a single column of a numeric, DATE, or DATETIME data type.

This type of distribution scheme is sometimes called a *range interval* distribution because:

- The RANGE and INTERVAL keywords are required in the DDL syntax that defines this strategy.
- The initial user-defined fragments are called *range fragments* to distinguish these fragments from system-defined fragments. The database server creates system-defined fragments automatically when a row is inserted whose fragment key value does not match the expression that defines any existing fragment.

The INTERVAL distribution strategy is useful when all possible fragment key values in a growing table are not known, and the DBA does not want to allocate fragments for data rows that are not yet loaded. For example, by using a DATE column as a fragment key could define a fragment for every month, or a BIGSERIAL column as a fragment key could define a fragment for every million customer records. The automatic creation of interval fragments avoids the need for a REMAINDER fragment (with its associated fragment-elimination difficulties) and can also reduce the maintenance workload of the DBA.

Defining an interval distribution strategy

The definition of an interval distribution scheme can include several required or optional parameters:

Fragment key

This must specify a column expression referencing a single numeric, DATE, or DATETIME column of the table.

Interval value expression

This constant expression defines an interval size within the range of fragment key values for system-generated interval fragments.

Storage location for interval fragments

This is a list of dbspaces where interval fragments will be stored.

Range fragment list

You must declare the name and define the fragment expression and the storage location for at least one range fragment.

The syntax for defining these parameters of a range interval distribution has this general form:

```
FRAGMENT BY RANGE (column_expr)
  INTERVAL (interval_size) STORE IN (dbspace_list)
  PARTITION partition VALUES < upper_bound IN dbspace,
  . . .
  PARTITION partition VALUES < upper_bound IN dbspace,
  PARTITION partition VALUES IS NULL IN dbspace
```

In this template, the syntax tokens that are not keywords specify the following parameters of the storage distribution scheme:

RANGE(*column_expr*)

This column expression, referencing a single column and delimited by parentheses, defines the fragment key. This clause is required.

INTERVAL(*interval_size*)

This interval value expression, delimited by parentheses, defines the interval size (within the range of values of the fragment key) for system-generated interval fragments. This clause is required.

STORE IN(*dbspace_list*)

This is a list of dbspaces where interval fragments will be stored. If you specify more than one dbspace, the database server creates successive new interval fragments in these dbspaces, in round robin fashion. This clause is optional.

If you omit this clause, the database server stores interval fragments in the dbspace that stores the range fragment. (If you define two or more range fragments and store them in different dbspaces, the database server stores each new interval fragment in one of these dbspaces, assigning successive interval fragments to these dbspaces in round robin fashion.)

PARTITION *partition*

This declares the name of a range or NULL fragment. You must define at least one range fragment. The NULL fragment is optional, but no more than one NULL fragment can be defined.

If you define more than one fragment, their names must conform to the rules for SQL identifiers, and must be unique among the fragments of the same table or index.

`VALUES < upper_bound`

This defines the fragment expression. Unlike list fragments, which can be defined in an arbitrary order, if you define more than one range fragment, their expressions must be defined in ascending order of the *upper_bound*. This clause is required.

The last range fragment that you define (which can be the first, if you define only one), is called the *transition fragment*, and its upper bound is called the *transition value* for the fragmented object. Any inserted rows with a larger fragment key value must be stored in an interval fragment.

`VALUES IS NULL`

This is the fragment expression to define the NULL fragment. Whether you define a NULL fragment is optional. The NULL fragment is not a range fragment, because NULL indicates the absence of a fragment key value. The database server issues an exception if a DML operation attempts to insert a row that has no fragment key value into a fragmented table for which no NULL fragment is defined.

If you define a NULL fragment, it can be listed in any position within the PARTITION specifications. The database server, rather than the sequence in which you declare the fragments, internally determines the order of each fragment within the fragment list of a table or index that uses an interval fragmentation scheme. The NULL fragment, if it exists, is always the first on this list, as indicated by its **sysfragments.evalpos** value in the system catalog.

When a row is inserted whose fragment key value is outside the range of any existing range or interval fragments, the database server will automatically create a new interval fragment based on the *interval_size* value and the transition value, without DBA intervention.

This kind of fragmentation strategy is useful when all possible fragment key values in a growing table are not known and the DBA does not want to allocate fragments for data that is not yet loaded.

Handle common dimensional data-modeling problems

The dimensional model that the previous sections describe illustrates only the most basic concepts and techniques of dimensional data modeling. The data model you build to address the business needs of your enterprise typically involves additional problems and difficulties that you must resolve to achieve the best possible query performance from your database. This section describes various methods you can use to resolve some of the most common problems that arise when you build a dimensional data model.

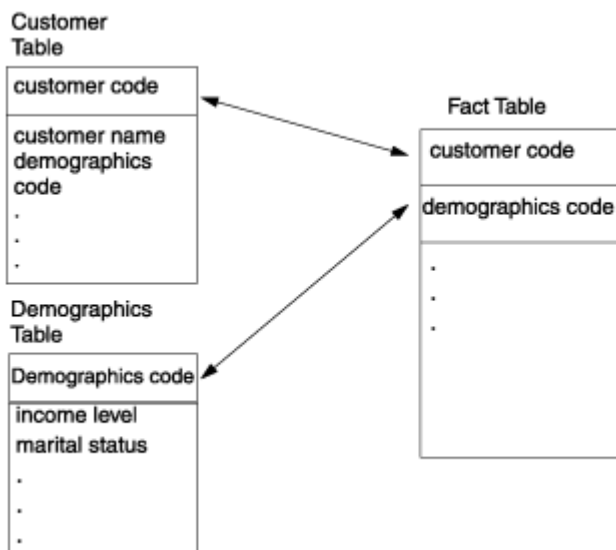
Minimize the number of attributes in a dimension table

Dimension tables that contain customer or product information might easily have 50 to 100 attributes and many millions of rows. However, dimension tables with too many attributes can lead to excessively wide rows and poor performance. For this reason, you might want to separate out certain groups of attributes from a dimension table and put them in a separate

table called a *minidimension* table. A minidimension table consists of a small group of attributes that are separated out from a larger dimension table. You might choose to create a minidimension table for attributes that have either of the following characteristics:

- The fields are rarely used as constraints in a query.
- The fields are frequently compared together.

The following figure shows a minidimension table for demographic information that is separated out from a **customer** table. Figure 10. A Minidimension Table for Demographics Information

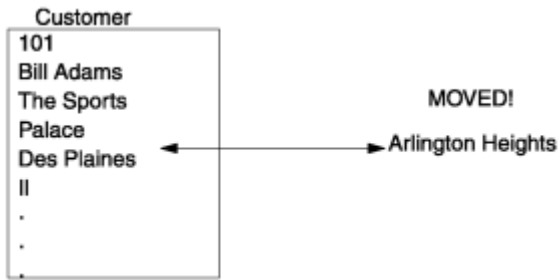


In the **demographics** table, you can store the demographics key as a foreign key in both the fact table and the **customer** table, which allows you to join the demographics table directly to the fact table. You can also use the demographics key directly with the **customer** table to browse demographic attributes.

Dimensions that occasionally change

In a dimensional database where updates are infrequent (as opposed to OLTP systems), most dimensions are relatively constant over time, because changes in sales districts or regions, or in company names and addresses, occur infrequently. However, to make historical comparisons, these changes must be handled when they do occur. The following figure shows an example of a dimension that has changed.

Figure 11. A dimension that changes



You can use three ways to handle changes that occur in a dimension:

Change the value stored in the dimension column

In the previous figure, the record for Bill Adams in the **customer** dimension table is updated to show the new address `Arlington Heights`. All of this customer's previous sales history is now associated with the district of Arlington Heights instead of Des Plaines.

Create a second dimension record with the new value and a generalized key

This approach effectively partitions history. The **customer** dimension table would now contain two records for Bill Adams. The old record with a key of 101 remains, and records in the fact table are still associated with it. A new record is also added to the **customer** dimension table for Bill Adams, with a new key that might consist of the old key plus some version digits (101.01, for example). All subsequent records that are added to the fact table for Bill Adams are associated with this new key.

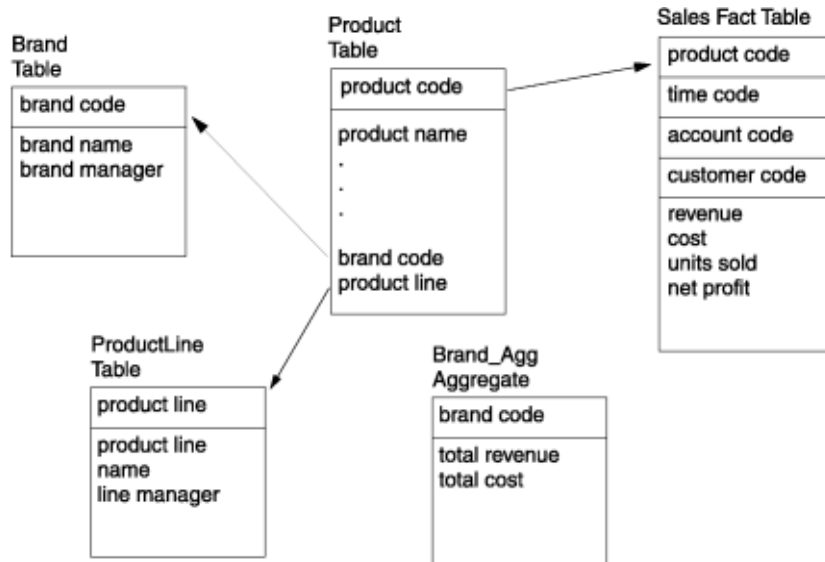
Add a new field in the customer dimension table for the affected attribute and rename the old attribute

This approach is rarely used unless you need to track old history in terms of the new value and vice-versa. The **customer** dimension table gets a new attribute named **current address**, and the old attribute is renamed **original address**. The record that contains information about Bill Adams includes values for both the original and current address.

Use the snowflake schema for hierarchical dimension tables

A *snowflake schema* is a variation on the star schema, in which very large dimension tables are normalized into multiple tables. Dimensions with hierarchies can be decomposed into a snowflake structure when you want to avoid joins to big dimension tables when you are using an aggregate of the fact table. For example, if you have brand information that you want to separate out from a **product** dimension table, you can create a brand snowflake that consists of a single row for each brand and that contains significantly fewer rows than the **product** dimension table. The following figure shows a snowflake structure for the brand and product line elements and the **brand_agg** aggregate table.

Figure 12. An example of a snowflake schema



If you create an aggregate table, **brand_agg**, that consists of the brand code and the total revenue per brand, you can use the snowflake schema to avoid the join to the much larger **sales** table. For example, you can use the following query on the **brand** and **brand_agg** tables:

```
SELECT brand.brand_name, brand_agg.total_revenue
FROM brand, brand_agg
WHERE brand.brand_code = brand_agg.brand_code
AND brand.brand_name = 'Anza'
```

Without a snowflaked dimension table, you use a `SELECT UNIQUE` or `SELECT DISTINCT` statement on the entire **product** table (potentially, a very large dimension table that includes all the brand and product-line attributes) to eliminate duplicate rows.

While snowflake schemas are unnecessary when the dimension tables are relatively small, a retail or mail-order business that has customer or product dimension tables that contain millions of rows can use snowflake schemas to significantly improve performance.

If an aggregate table is not available, any joins to a dimension element that was normalized with a snowflake schema must now be a three-way join, as the following query shows. A three-way join reduces some of the performance advantages of a dimensional database.

```
SELECT brand.brand_name, SUM(sales.revenue)
FROM product, brand, sales
WHERE product.brand_code = brand.brand_code
AND brand.brand_name = 'Alltemp'
GROUP BY brand_name
```

Related information

Keys to join the fact table with the dimension tables

Implement a dimensional database

You will learn the SQL statements that you need to implement the dimensional data model

This section shows you the SQL statements required to implement the dimensional database that is described in the section [Design a dimensional data model on page 7](#). Remember that this database serves only as an illustrative example of a data-warehousing environment. For the sake of the example, it is translated into SQL statements.

This section describes the **sales_demo** database.

Related reference

[Design a dimensional data model on page 7](#)

Implement the sales_demo dimensional database

This section shows the SQL statements that you can use to create a dimensional database from the data model that you learned about in [Design a dimensional data model on page 7](#). You can use interactive SQL to write the individual statements that create the database or you can run a script that automatically executes all the statements that you need to implement the database. The CREATE DATABASE and CREATE TABLE statements create the data model as tables in a database. After you create the database, you can use the LOAD and INSERT statements to populate the tables.

Create the dimensional database

You must create the dimensional database before you can create any of the tables or other objects that the database must contain.

When you use the Informix® database server to create a database, the server sets up records that show the existence of the database and its mode of logging. The database server manages disk space directly, so these records are not visible to operating-system commands.

The following statement shows the syntax that you use to create a database that is called **sales_demo**:


```
CREATE DATABASE sales_demo
```

The CREATE TABLE statement for the dimension and fact tables

This section includes the CREATE TABLE statements that you use to create the tables of the **sales_demo** dimensional database.

Referential integrity is, of course, an important requirement for dimensional databases. However, the following schema for the **sales_demo** database does not define the primary and foreign key relationships that exist between the fact table and its dimension tables. The schema does not define these primary and foreign key relationships because data-loading performance improves dramatically when the database server does not enforce constraint checking. Given that data

warehousing environments often require that tens or hundreds of gigabytes of data are loaded within a specified time, data-load performance should be a factor when you decide how to implement a database in a warehousing environment. Assume that if the **sales_demo** database is implemented as a live data mart, some data extraction tool (rather than the database server) is used to enforce referential integrity between the fact table and dimension tables.

 **Tip:** After you create and load a table, you can add primary key and foreign key constraints to the table with the ALTER TABLE statement to enforce referential integrity. This method is required only for express load mode. If the constraints and indexes are necessary and costly to drop before a load, then deluxe load mode is the best option.

The following statements create the **time**, **geography**, **product**, and **customer** tables. These tables are the dimensions for the **sales** fact table. A SERIAL field serves as the primary key for the **district_code** column of the **geography** table.

```
CREATE TABLE time
(
time_code      INT,
order_date     DATE,
month_code     SMALLINT,
month_name     CHAR(10),
quarter_code   SMALLINT,
quarter_name   CHAR(10),
year           INTEGER
);

CREATE TABLE geography
(
district_code  SERIAL,
district_name  CHAR(15),
state_code     CHAR(2),
state_name     CHAR(18),
region        SMALLINT
);

CREATE TABLE product (
product_code   INTEGER,
product_name   CHAR(31),
vendor_code    CHAR(3),
vendor_name    CHAR(15),
product_line_code SMALLINT,
product_line_name CHAR(15)
);

CREATE TABLE customer (
customer_code  INTEGER,
customer_name  CHAR(31),
company_name   CHAR(20)
);
```

The **sales** fact table has pointers to each dimension table. For example, **customer_code** references the customer table, **district_code** references the geography table, and so forth. The **sales** table also contains the measures for the units sold, revenue, cost, and net profit.

```
CREATE TABLE sales
(
customer_code  INTEGER,
```

```

district_code SMALLINT,
time_code     INTEGER,
product_code  INTEGER,
units_sold   SMALLINT,
revenue      MONEY(8,2),
cost         MONEY(8,2),
net_profit   MONEY(8,2)
);

```

i **Tip:** The most useful measures (facts) are numeric and additive. Because of the great size of databases in data-warehousing environments, virtually every query against the fact table might require thousands or millions of records to construct a result set. The only useful way to compress these records is to aggregate them. In the **sales** table, each column for the measures is defined on a numeric data type, so you can easily build result sets from the **units_sold**, **revenue**, **cost**, and **net_profit** columns.

For your convenience, the file called `createdw.sql` contains all the preceding CREATE TABLE statements.

Mapping data from data sources to the database

The **stores_demo** demonstration database is the primary data source for the **sales_demo** database.

The following table shows the relationship between data warehousing business terms and the data sources. It also shows the data source for each column and table of the **sales_demo** database.

Table 7. The relationship between data warehousing business terms and data sources

Business Term	Data Source	Table.Column Name
Sales Fact Table:		
product code		sales.product_code
customer code		sales.customer_code
district code		sales.district_code
time code		sales.time_code
revenue	stores_demo:items.total_price	sales.revenue
units sold	stores_demo:items.quantity	sales.units_sold
cost	costs.lst (per unit)	sales.cost
net profit	calculated: revenue minus cost	sales.net_profit
Product Dimension Table:		
product	stores_demo:catalog.catalog_num	product.product_code

Table 7. The relationship between data warehousing business terms and data sources**(continued)**

Business Term	Data Source	Table.Column Name
product name	stores_demo:stock.manu_code and stores_demo:stock.description	product.product_name
product line	stores_demo:orders.stock_num	product.product_line_code
product line name	stores_demo:stock.description	product.product_line_name
vendor	stores_demo:orders.manu_code	product.vendor_code
vendor name	stores_demo:manufact.manu_name	product.vendor_name
Customer Dimension Table:		
customer	stores_demo:orders.customer_num	customer.customer_code
customer name	stores_demo:customer.fname plus stores_demo:customer.lname	customer.customer_name
company	stores_demo:customer.company	customer.company_name
Geography Dimension Table:		
district code	generated	geography.district_code
district	stores_demo:customer.city	geography.district_name
state	stores_demo:customer.state	geography.state_code
state name	stores_demo.state.sname	geography.state_name
region	derived: If state = "CA" THEN region = 1, ELSE region = 2	geography.region
Time Dimension Table:		
time code	generated	time.time_code
order date	stores_demo:orders.order_date	time.order_date
month	derived from order date generated	time.month_name time.month_code
quarter	derived from order date generated	time.quarter_name time.quarter_code
year	derived from order date	time.year


Several files with a `.unl` suffix contain the data that is loaded into the **sales_demo** database. The files that contain the SQL statements that create and load the database have a `.sql` suffix.

If your database server runs on UNIX™, you can access the `*.sql` and `*.unl` files from the directory `$INFORMIXDIR/demo/dbaccess`.

If your database server runs on Windows™, you can access the *.sql and *.unl files from the directory %INFORMIXDIR%\demo\dbaccess.

Load data into the dimensional database

An important step when you implement a dimensional database is to develop and document a load strategy. This section shows the LOAD and INSERT statements that you can use to populate the tables of the **sales_demo** database.

 **Tip:** In a live data warehousing environment, you typically do not use the LOAD or INSERT statements to load large amounts of data to and from Informix® databases.

Informix® database servers provide different features for loading and unloading of data.

For information about loading, see your *Informix® Administrator's Guide*.

The following statement loads the **time** table with data first so that you can use it to determine the time code for each row that is loaded into the **sales** table:

```
LOAD FROM 'time.unl' INSERT INTO time
```

The following statement loads the **geography** table. After you load the **geography** table, you can use the district code data to load the **sales** table.

```
INSERT INTO geography(district_name, state_code, state_name)
SELECT DISTINCT c.city, s.code, s.sname
  FROM stores_demo:customer c, stores_demo:state s
   WHERE c.state = s.code
```

The following statements add the region code to the **geography** table:

```
UPDATE geography
  SET region = 1
  WHERE state_code = 'CA'

UPDATE geography
  SET region = 2
  WHERE state_code <> 'CA'
```

The following statement loads the **customer** table:

```
INSERT INTO customer (customer_code, customer_name, company_name)
SELECT c.customer_num, trim(c.fname) || ' ' || c.lname, c.company
  FROM stores_demo:customer c
```

The following statement loads the **product** table:

```
INSERT INTO product (product_code, product_name, vendor_code,
  vendor_name, product_line_code, product_line_name)
SELECT a.catalog_num,
  trim(m.manu_name) || ' ' || s.description,
  m.manu_code, m.manu_name,
  s.stock_num, s.description
  FROM stores_demo:catalog a, stores_demo:manufact m,
  stores_demo:stock s
```



```
WHERE a.stock_num = s.stock_num
      AND a.manu_code = s.manu_code
      AND s.manu_code = m.manu_code;
```

The following statement loads the **sales** fact table with one row for each product, per customer, per day, per district. The cost from the **cost** table is used to calculate the total cost (cost * quantity).

```
INSERT INTO sales (customer_code, district_code, time_code,
                  product_code, units_sold, cost, revenue, net_profit)
SELECT
  c.customer_num, g.district_code, t.time_code,
  p.product_code, SUM(i.quantity),
  SUM(i.quantity * x.cost), SUM(i.total_price),
  SUM(i.total_price) - SUM(i.quantity * x.cost)
FROM stores_demo:customer c, geography g, time t,
     product p, stores_demo:items i,
     stores_demo:orders o, cost x
WHERE c.customer_num = o.customer_num
      AND o.order_num = i.order_num
      AND p.product_line_code = i.stock_num
      AND p.vendor_code = i.manu_code
      AND t.order_date = o.order_date
      AND p.product_code = x.product_code
      AND c.city = g.district_name
GROUP BY 1,2,3,4;
```

Test the dimensional database

After you create the tables and load the data into the database, you should test the dimensional database.

You can create SQL queries to retrieve the data necessary for the standard reports listed in the business-process summary (see the [Summary of a business process on page 13](#)). Use the following ad hoc queries to test that the dimensional database was properly implemented.

The following statement returns the monthly revenue, cost, and net profit by product line for each vendor:

```
SELECT vendor_name, product_line_name, month_name,
       SUM(revenue) total_revenue, SUM(cost) total_cost,
       SUM(net_profit) total_profit
FROM product, time, sales
WHERE product.product_code = sales.product_code
      AND time.time_code = sales.time_code
GROUP BY vendor_name, product_line_name, month_name
ORDER BY vendor_name, product_line_name;
```

The following statement returns the revenue and units sold by product, by region, and by month:

```
SELECT product_name, region, month_name,
       SUM(revenue), SUM(units_sold)
FROM product, geography, time, sales
WHERE product.product_code = sales.product_code
      AND geography.district_code = sales.district_code
      AND time.time_code = sales.time_code
GROUP BY product_name, region, month_name
ORDER BY product_name, region;
```

The following statement returns the monthly customer revenue:

```
SELECT customer_name, company_name, month_name,
       SUM(revenue)
FROM customer, time, sales
WHERE customer.customer_code = sales.customer_code
      AND time.time_code = sales.time_code
GROUP BY customer_name, company_name, month_name
ORDER BY customer_name;
```

The following statement returns the quarterly revenue per vendor:

```
SELECT vendor_name, year, quarter_name, SUM(revenue)
FROM product, time, sales
WHERE product.product_code = sales.product_code
      AND time.time_code = sales.time_code
GROUP BY vendor_name, year, quarter_name
ORDER BY vendor_name, year
```

Change the storage distribution strategy

Use the ALTER FRAGMENT statement to change the storage distribution strategy of the data rows that are being loaded into an existing database table.

You should adjust the storage distribution strategy when the volume or distribution of the data is different than what was originally expected when the storage distribution plan was first implemented. The ALTER FRAGMENT can also be used as part of the workflow of a data warehouse. If a table is partitioned with a fragment key that is based on values in a DATE or DATETIME column, the fragments can be detached from the table. As new fragments are added to the table older fragments, that store rows from earlier time periods, can be detached from the table.

The ALTER FRAGMENT statement supports the following six options for table fragments. Some ALTER FRAGMENT options are valid for nonfragmented tables or for index fragments.



Note: The following summary ignores tables that are fragmented by ROUND ROBIN, because other fragmentation strategies are more often used in data warehousing applications.

ADD

Adds a new fragment in the list of fragments that are part of a table that has been fragmented by any fragmentation scheme.

For LIST or EXPRESSION fragments, you can add a NULL fragment or a REMAINDER fragment, if none of these types of fragments have already been defined. You can use the BEFORE or AFTER keyword to specify the ordinal position of the new fragment in the list of expressions or list of fragments.

For a table that has been fragmented with the INTERVAL option, you can use the ADD option can add new storage spaces to the list of dbspaces where the database server creates new INTERVAL fragments.

ATTACH

Combines two or more tables that have identical structures into a fragmentation strategy. All of the consumed tables specified in the ALTER FRAGMENT ATTACH statement must have the same structure as the surviving table. The number, names, data types, and relative positions of the columns must be identical. The consumed tables must be nonfragmented, and must be stored in a different dbspace from the surviving table. The ATTACH option does not support index fragments.

For LIST or EXPRESSION fragments, you can attach a NULL fragment or a REMAINDER fragment, if none of these types of fragments have already been defined. You can use the BEFORE or AFTER keyword to specify the ordinal position of the new fragment in the list of fragments.

For a table that has been fragmented by INTERVAL, the ATTACH option can attach new RANGE fragments. However you cannot attach new INTERVAL fragments, and you cannot use the BEFORE or AFTER keyword to specify the ordinal position of the new RANGE fragment.

When a new EXPRESSION fragment is attached to table that is fragmented by LIST or by INTERVAL, the rows from the consumed table and the affected fragments in the surviving table are scanned and moved into appropriate partitions. These strategies are not overlapping.

You can also include the ONLINE keyword in ALTER FRAGMENT ATTACH statements with interval partitioning. Specifying this keyword can improve concurrency for other sessions that attempt to access the tables on which the ALTER FRAGMENT ONLINE statement operates.

DETACH

Removes a table fragment from a distribution scheme and places the contents into a new, nonfragmented table. The DETACH option does not support index fragments.

The table from which the fragment was detached remains fragmented, unless it is fragmented by LIST or by EXPRESSION and had only two fragments before the DETACH operation.

The new table does not inherit any indexes, constraints, or discretionary access privileges of the table from which it was detached. The new table has the default access privileges of any new table.

The ALTER FRAGMENT DETACH statement cannot remove a fragment from a table that is the parent of a referential constraint, or from a table on which a ROWID column is defined.

You can also include the ONLINE keyword in ALTER FRAGMENT DETACH statements with interval partitioning.

DROP

Drops a fragment from a table or index that is fragmented by LIST or by EXPRESSION. Using the DROP option requires, however, that any rows currently stored in the fragment can be moved to another existing fragment. For LIST fragments, the existing fragment can only be the REMAINDER fragment, because of the uniqueness requirement for LIST fragment expressions.

For a table or index that is fragmented by INTERVAL, you can use the DROP option to drop one or more dbspaces from the list of dbspaces that store INTERVAL fragments. No new INTERVAL fragments will be created in the specified dbspaces.

ALTER TABLE DROP operations that result in moving a large number of rows can fail if insufficient log space or disk space is available. You might be able to complete the operation by dividing it into a series of smaller operations. If insufficient log space causes the failure, an alternative is to temporarily turning off logging. Then retry the ALTER TABLE operation and turn transaction-logging back on after the operation completes. To perform ALTER TABLE DROP operations requires a backup of the **root** dbspace.

INIT

Defines, modifies, or replaces the fragmentation strategy or the storage location of an existing table or an existing index.

For an index, you can accomplish these tasks:

- Change an existing fragmented index to a nonfragmented index.
- Change the interval value of the interval distribution scheme for a fragmented index.
- Change the interval fragment key of the interval distribution scheme for a fragmented index.
- Fragment an existing index that is not fragmented without redefining the index.
- Change the distribution scheme of an existing fragmented index to another distribution scheme of the same expression, list, or interval type, or to a different type of distribution scheme.

For a nonfragmented table, you can accomplish these tasks:

- Move a nonfragmented table from one dbspace to another dbspace.
- Move a nonfragmented table from one dbspace to a named fragment.
- Change a nonfragmented table to a fragmented table.

For a fragmented table, you can accomplish these tasks:

- Convert a fragmented table to a nonfragmented table.
- Replace the current fragmentation scheme with a different fragmentation scheme of the same type or of a different type
- Change the expression associated with an existing list-based or expression-based fragment
- Add a new rowid column to a fragmented table. This column stores a unique integer that cannot be updated. The database server automatically creates an index on the rowid column. With this column, the database server can find the physical location of any row.

If the table is not empty when you convert an existing storage fragmentation strategy to another strategy, the database server discards the existing fragmentation strategy and moves data rows to fragments that you define in the new fragmentation strategy. Data movement also occurs when you convert a nonfragmented index to a fragmented index, and when you convert a fragmented index to a nonfragmented index. For large tables, data movement can cause significant logging, or the transaction might approach the long-transaction high-watermark, and a relatively long exclusive lock might be held on the affected tables. Use these ALTER FRAGMENT INIT options when they do not interfere with day-to-day operations.

MODIFY

Change the current fragmentation list of a table or of an index.

For LIST or EXPRESSION fragments, the MODIFY option can accomplish these tasks:

- Move an existing fragment from one dbspace to a different dbspace.
- Change the expression associated with an existing list-based or expression-based fragment.
- Rename one or more existing fragments.

For a table that is fragmented by INTERVAL, the MODIFY option can accomplish these tasks:

- Modify the expression that defines a range fragment.
- Increase the value of the expression that defines the transition value of the table.
- Enable or disable the automatic creation of interval fragments.
- Replace the list of dbspaces where system-generated interval fragments will be created. Existing fragments in the old dbspaces are not moved, and new rows that match their fragment expressions will be inserted into those fragments.
- Move a range fragment or an interval fragment to a different dbspace.
- Rename one or more existing fragments.

When you change the expression that defines a range fragment, the replacement expression cannot cross adjacent fragment boundaries.

You cannot modify the system-generated expression for any INTERVAL fragment, and you cannot decrease the transition value of a table that is fragmented by INTERVAL.

You can also include the ONLINE keyword in ALTER FRAGMENT ON TABLE INTERVAL TRANSITION statements.

Moving data from relational tables into dimensional tables by using external tables

Use SQL statements to unload data from relational tables into external tables, which are data files that are in table format, and then load the data from the data files into the dimensional tables.

Before you begin

Before beginning, document a strategy for mapping data in the relational database to the dimensional database.

About this task

To unload data from the relational database into external tables and then load the data into the dimensional database:

1. Unload the data from a relational database to external tables.

Repeat the following steps to create as many external tables as are required for the data that you want to move.

- a. Use the CREATE EXTERNAL TABLE statement to describe the location of the external table and the format of the data.

Example

The following sample CREATE EXTERNAL TABLE statement creates an external table called `emp_ext`, with data stored in a specified fixed format:

```
CREATE EXTERNAL TABLE emp_ext
( name CHAR(18) EXTERNAL CHAR(18),
  hiredate DATE EXTERNAL CHAR(10),
  address VARCHAR(40) EXTERNAL CHAR(40),
  empno INTEGER EXTERNAL CHAR(6) )
USING (
  FORMAT 'FIXED',
  DATAFILES
  ("DISK:/work2/mydir/employee.unl")
);
```

- b. Use the INSERT...SELECT statement to map the relational database table to the external table.

Example

The following sample INSERT statement loads the employee database table into the external table called emp_ext:

```
INSERT INTO emp_ext SELECT * FROM employee
```

The data from the employee database table is stored in a data file called employee.unl.

2. If necessary, copy or move the data files to the system where the dimensional database is located.
3. Load the data from the data files to the dimensional database.

Repeat the following steps to load all the data files that you created in the previous steps.

- a. Use the CREATE EXTERNAL TABLE statement to describe the location of the data file and the format of the data.

Example

The following code is a sample CREATE EXTERNAL TABLE statement:

```
CREATE EXTERNAL TABLE emp_ext
( name CHAR(18) EXTERNAL CHAR(18),
  hiredate DATE EXTERNAL CHAR(10),
  address VARCHAR(40) EXTERNAL CHAR(40),
  empno INTEGER EXTERNAL CHAR(6) )
USING (
  FORMAT 'FIXED',
  DATAFILES
  ("DISK:/work3/mydir/employee.unl")
);
```

- b. Use the INSERT...SELECT statement to map the data from the data file to the table in the dimensional database.

Example

The following sample INSERT statement loads the employee data file into the employee database table:

```
INSERT INTO employee SELECT * FROM emp_ext
```

Performance tuning dimensional databases

This section describes how to tune the performance of your queries and to understand data distribution statistics.

Related reference

[Design a dimensional data model on page 7](#)

Query execution plans

When a SELECT statement or other DML operation is submitted to the database server, the query execution optimizer designs a query execution plan. The query execution optimizer is often referenced as the *query optimizer*.

To design a query execution plan, and estimate the costs of candidate query plans, the query optimizer considers a wide range of information including:

- Specifications that identify the database objects, predicates, filters, joins, and other operations in the SQL syntax that defines the query operation
- System catalog information about indexes and constraints on the tables, views, and columns that are referenced or implied in the query
- Data distribution statistics for the tables and indexes, or for their fragments, that are referenced or implied in the query
- Optimizer directives that are specified inline or as external optimizer directives that favor or avoid subsets of the potential query plans
- Information in the database server environment or in the session environment that affects the query execution optimizer

Data distribution statistics

Data distribution statistics are stored in the system catalog for use by the query optimizer when it designs query execution plans. These statistics, together with other information, enable the optimizer to estimate the relative costs among the execution plans that the optimizer is considering for a specific query. Distribution statistics that the optimizer examines for tables that are referenced in queries can include column distribution statistics for the table and for its indexes, as well as fragment-level statistics, if the database server has gathered statistics for individual table or index fragments.

The following system catalog tables store data distribution information that is available to the query optimizer:

SYSDISTRIB

Stores data distribution information for tables and indexes.

SYSFRAGDIST

Stores fragment-level data distribution information for fragments of tables and indexes.

The following system catalog tables store information pertaining to changes to rows since the most recent update to table, index, or fragment statistics.

SYSDISTRIB

Counts the number of rows changed by DML operations since table statistics were last updated, the date and time of that update, and the time required to build column distributions.

SYSPRAGDIST

Counts the number of rows changed by DML operations since fragment-level statistics were last updated, and the date and time of that update.

SYSPRAGMENTS

Counts the number of rows changed by DML operations since fragment-level statistics were last updated.

SYSPINDICES

Counts the number of rows changed by DML operations since index statistics were last updated, the date and time of that update, and time required to build low level distributions for the lead column of the index.

The following configuration parameters can affect the database server behavior for the calculation, display, or other operations on data distribution statistics for tables or for fragments that can be used in query plans:

AUTO_STAT_MODE

Enable or disable the detection (and selective refreshing) of stale statistics during UPDATE STATISTICS operations. You can override the setting of this parameter by using the **onmode -wm** or **onmode -wf** command-line utilities, or SQL administration API function calls, or (for the current session) by the SET ENVIRONMENT AUTO_STAT_MODE statement of SQL.

EXPLAIN_STAT

Enable or disable the inclusion of a Query Statistics section in the explain output file. This is enabled by default.

SYSSBSPACENAME

Specifies the name of the sbspace in which the database server stores data-distribution statistics (as smart large objects) that the UPDATE STATISTICS statement collects for certain user-defined data types. Because the data distributions for UDTs can be large, you have the option to store them in an sbspace instead of in the **sysdistrib** system catalog table (for table-level statistics) or in the **sysfragdist** system catalog table (for fragment-level statistics), where distribution statistics are stored by default.

STATCHANGE

Specifies a positive integer as a change threshold to identify table or fragment distribution statistics that need to be updated. This is the default threshold for refreshing distribution statistics on tables for which no specific threshold has been specified as a table or session attribute. If no value is specified, the default is 10. While selective refreshing of data distribution statistics enabled (by default, or by the AUTO_STAT_MODE setting, or by the AUTO keyword of the UPDATE STATISTICS statement, UPDATE STATISTICS operations only refresh stale or missing statistics. The default value of 10 restricts recalculation to only the tables or fragments in which DML, load, or TRUNCATE operations have changed more than 10% of the rows since data distribution statistics were most recently calculated.

Fragment-level statistics

For tables and indexes that have been partitioned according to fragment key values, the distribution statistics in the system catalog for some fragments might closely approximate current data distributions in those fragments, despite subsequent DELETE, INSERT, UPDATE, or MERGE operations that have caused the statistics for other fragments to become stale. For large tables that contain millions of rows, substantial resources of the database server can be conserved by updating only the subset of fragments with stale statistics, rather than recalculating distribution statistics for every fragment.

The STATLEVEL table attribute

For tables and indexes that are partitioned into multiple fragments by a distributed storage scheme, you can specify the granularity of its data distribution statistics, and you can specify the criteria by which stale statistics are defined. This can be accomplished by specifying keyword options of the Statistics Options clause in either of two DDL statements:

- in the CREATE TABLE statement (when defining a new fragmented table)
- in the ALTER TABLE statement (when changing the statistics granularity of an existing fragmented table).

In both cases, your options for specifying the granularity of the distribution statistics are the same:

STATLEVEL AUTO

Specifies that the database server apply the following criteria at runtime to determine if fragment-level distributions should be created:

- The table is fragmented by EXPRESSION, by LIST, or by INTERVAL.
- The table has more than 1,000,000 rows.

Unless both of these criteria are satisfied, table-level distributions are created. AUTO is the default setting in the CREATE TABLE statement, if you specify no explicit STATLEVEL setting.

STATLEVEL FRAGMENT

Data distributions will be created and maintained for each fragment. The FRAGMENT option is not valid for nonfragmented tables, or for tables that use a round robin storage distribution scheme.

STATLEVEL TABLE

All data distributions for the table will be created at the table level. This emulates the legacy behavior of Informix® servers earlier than version 11.70.

To support fragment level data distribution statistics, you must specify the name of an sbspace as the setting of the SYSSBSPACENAME configuration parameter, and you must also declare the name and allocate storage for that sbspace by using the **-c -S** option of the **onspaces** utility. For any table whose STATLEVEL attribute is set to FRAGMENT, the database server returns an error if SYSSBSPACENAME is not set, or if the sbspace to which is SYSSBSPACENAME is set is not properly allocated. For each fragment, the most recently calculated data distribution statistics are stored as a BLOB object in the **sysfragdist.ncndist** column in the system catalog.

Data distribution statistics gathered at the fragment level can be aggregated to provide table level statistics from the constituent fragment statistics.

The STATCHANGE threshold for refreshing data distribution statistics

The same Statistics Options clause of the CREATE TABLE or ALTER TABLE statement can also specify a change threshold for data distribution statistics. The database server applies this STATCHANGE attribute of a fragmented table to all of the fragments of the table. The STATCHANGE table attribute can be set to an integer value, or you can specify the AUTO keyword:

integer

This defines an integer change threshold between 0 and 100 which defines how much table or fragment data is allowed to change before its statistics are considered stale in UPDATE STATISTICS operations that selectively update only stale distribution statistics.

AUTO

The threshold is the value of the STATCHANGE configuration parameter (or else 10, if no value is set for the STATCHANGE parameter). If the SET ENVIRONMENT statement has set a different value for the current session, that value overrides the default or explicit STATCHANGE configuration parameter setting.

AUTO is the default setting in the CREATE TABLE statement, if you specify no explicit STATCHANGE setting.

For the table and index fragments for which data distribution statistics are already stored in the system catalog, the STATCHANGE setting specifies the percentage of rows in the fragment that have been deleted, inserted, or modified by DML operations since its distribution statistics were most recently updated. (This is the same significance that STATCHANGE has for table-level statistics.)

Automatic management of data distribution statistics

The Informix® database server supports several mechanisms for automating some of the tasks that are involved in gathering, dropping, and refreshing data distribution statistics for tables, indexes, table fragments, and index fragments.

Automatic detection and refreshing of stale statistics during UPDATE STATISTICS operations

You can set the AUTO_STAT_MODE configuration parameter to enable the Informix® database server to automatically detect which table and index statistics are stale, and only refresh the stale statistics when the UPDATE STATISTICS statement is run. The data distribution statistics that are automatically detected and refreshed are calculated at the table, fragment, or index level, not at the individual column level. If you set no value for this parameter, the automatic statistics mode is enabled by default. When automatic mode is enabled, the default threshold that defines stale statistics is reached when at least 10% of the rows in the table or fragment are changed by DML, LOAD, or TRUNCATE operations since the most recent calculation of data distribution statistics.

You can set another configuration parameter, STATCHANGE, to specify a nondefault change threshold for refreshing distribution statistics when automatic statistics mode is enabled. For example, if you set the STATCHANGE value to 15, statistics are refreshed if 15% of the rows in the table or fragment are changed. If the STATCHANGE parameter is not set, the system default value for STATCHANGE is 10.

You can override the `STATCHANGE` or `AUTO_STAT_MODE` configuration parameter setting for the current session by using the `SET ENVIRONMENT` statement to set session environment variables of the same names. The DBA can include `SET ENVIRONMENT` statements in the **`sysdbopen`** routine to enable or disable automatic statistics mode, or to change the stale distribution threshold (or both) at connection time. These settings are applied to `UPDATE STATISTICS` statements that are issued in the current session.

A table can be created with its own `STATCHANGE` table attribute, whose value overrides the setting of the `STATCHANGE` session environment variable or configuration parameter. For fragmented tables whose distribution statistics are calculated for each fragment, the value of its `STATCHANGE` attribute determines whether statistics are refreshed for individual fragments. The `ALTER TABLE` statement of SQL can reset the `STATCHANGE` attribute of a table.

You can also use (or disable for your current operation) the explicit or default `AUTO_STAT_MODE` and `STATCHANGE` settings during `UPDATE STATISTICS` statements that include the `AUTO` or the `FORCE` keyword:

AUTO

This keyword puts the `UPDATE STATISTICS` statement in automatic mode for detecting tables and fragments whose statistics are stale. Distribution statistics are not refreshed for tables or fragments whose `STATCHANGE` value is below the specified threshold.

FORCE

This keyword refreshes the statistics for all tables and columns within the specified scope. If automatic mode for detecting stale statistics is enabled, the `FORCE` keyword overrides automatic mode, so that values of the `STATCHANGE` attributes of tables and fragments are ignored, and statistics are recalculated for all database objects within the scope of the `FOR TABLE` specification.

The scope of `AUTO` or `FORCE` is limited to the `UPDATE STATISTICS` statement in which the keyword is specified. `UPDATE STATISTICS` statements that include neither of these keywords use the current `AUTO_STAT_MODE` setting of the database server (or for their session environment, if that is different). If `AUTO_STAT_MODE` is enabled, the `STATCHANGE` value is determined in the following (descending) order of precedence:

1. The value of the `STATCHANGE` attribute of the table, if `AUTO` is not the specified value.
2. The value that is set by the most recent `SET ENVIRONMENT STATCHANGE` statement in the same session.
3. The explicit setting of the `STATCHANGE` configuration parameter.
4. The system default `STATCHANGE` value is 10.

Automatic statistics maintenance in DDL operations

The Informix® database server automatically creates, updates, or drops data distribution statistics during certain operations that create, alter, or destroy database objects.

ALTER FRAGMENT ATTACH operations

If the automatic mode for detecting stale distribution statistics is enabled, and the table that is being attached to has fragmented distribution statistics, the database server calculates the distribution statistics of the new fragment. Stale distribution statistics of existing fragments are also recalculated. This recalculation of fragment statistics runs in the background. After the database server calculates the fragment statistics, it merges them to form table distribution statistics, and stores the results in the system catalog.

Distribution statistics are not recalculated, however, unless explicit or default value of the `AUTO_STAT_MODE` configuration parameter or the `AUTO_STAT_MODE` session environment setting enables the automatic mode for detecting stale data distribution statistics.

ALTER FRAGMENT DETACH operations

Some `ALTER FRAGMENT DETACH` statements to attach a fragment can cause the database server to update the index structure. When an index is rebuilt in those cases, the database server also recalculates the associated column distributions, and these statistics are available to the query optimizer when it designs query plans for the table from which the fragment was detached:

- For an indexed column (or for a set of columns) on which `ALTER FRAGMENT DETACH` automatically rebuilds a B-tree index, the recalculated column distribution statistics are equivalent to distributions created by the `UPDATE STATISTICS` statement in `HIGH` mode.
- If the rebuilt index is not a B-tree index, the automatically recalculated statistics correspond to distributions created by the `UPDATE STATISTICS` statement in `LOW` mode.

If the automatic mode for detecting stale distribution statistics is enabled, and the table from which the fragment is being detached has fragment-level distribution statistics, the database server takes the following actions:

- Uses the distribution statistics of the detached fragment to form a new table distribution.
- Merges the distribution statistics of the remaining fragments to calculate distribution statistics for the surviving table
- Stores the statistics that result from these operations in the system catalog.

This recalculation of fragment statistics runs in the background.

Distribution statistics are not recalculated, however, unless an explicit or default value of the `AUTO_STAT_MODE` configuration parameter or the `AUTO_STAT_MODE` environment setting enables the automatic mode for detecting stale data distribution statistics.

ALTER TABLE ADD CONSTRAINT operations

`ALTER TABLE ADD CONSTRAINT` statements that use the Single Column Constraint format to implicitly create an index on a non-opaque column also automatically calculate the distribution of the specified column. Similarly, if the Multiple-Column Constraint format specifies a list of columns as the scope of the new constraint, the database server implicitly creates an index on the same non-opaque column or set of columns as the referential constraint, distribution statistics are automatically calculated on the specified column, or on the lead column of a multiple-column constraint.

These distribution statistics are available to the query optimizer when it designs query plans for the table on which the constraint is defined:

- For columns on which the new constraint is implemented as a B-tree index, the recalculated column distribution statistics are equivalent to distributions created by the UPDATE STATISTICS statement in HIGH mode.
- If the new constraint is not implemented as a B-tree index, the automatically recalculated statistics correspond to distributions created by the UPDATE STATISTICS statement in LOW mode.

These distribution statistics are available to the query optimizer when it designs query plans for the table on which the new constraint was created.

The automatic calculation of column distribution statistics in ALTER TABLE MODIFY operations that define a constraint on a non-opaque column is not dependent on whether AUTO_STAT_MODE is enabled or disabled.

ALTER TABLE MODIFY operations

ALTER TABLE MODIFY statements that use the Single Column Constraint format or Multiple Column Constraint format to define constraints similarly cause the database server to calculate data distribution statistics for the indexes that are implicitly created to enforce the constraints. These distribution statistics have the same attributes as the statistics that are calculated automatically for an index on a non-opaque column, and that are also automatically calculated during ALTER TABLE ADD CONSTRAINT operations. These statistics are available to the query optimizer when it designs query plans for the table on which the constraints are defined.

The automatic calculation of column distribution statistics in ALTER TABLE MODIFY operations that define a constraint on a non-opaque column is not dependent on whether AUTO_STAT_MODE is enabled or disabled.

CREATE INDEX operations

The database server automatically calculates index statistics, equivalent to the statistics gathered by UPDATE STATISTICS in LOW mode, when you create a B-tree index on a UDT column of an existing table, or if you create a functional index or a virtual index interface (VII) index on a column of an existing table. Statistics that are collected automatically by this feature are stored in the system catalog and are available to the query optimizer, without the need for running the UPDATE STATISTICS statement manually. When B-tree indexes are created, column statistics are collected on the first index column, equivalent to what UPDATE STATISTICS generates in HIGH mode, with a resolution is 1% for tables of fewer than a million rows, and 0.5% for larger tables. (Tables with more than 1 million rows have a better resolution because they have more bins for statistics.)

The automatic calculation of column distribution statistics in CREATE INDEX operations is not dependent on whether AUTO_STAT_MODE is enabled or disabled.

Auto Update Statistics (AUS) maintenance system

This uses a combination of Scheduler sensors, tasks, thresholds, and tables to evaluate and update data distribution statistics. The system provides as built-in input criteria a set of configuration parameter values. The system administrator can modify these to reflect current requirements and workloads. The AUS system combines these criteria with information from the **sysmaster** database to automatically identify tables whose distributions are becoming stale, and generates the text of UPDATE STATISTICS statements to refresh the distribution statistics for those tables.

The list of generated UPDATE STATISTICS statements is run automatically each week at a designated period of low throughput, to update as many table distributions as can be recalculated during the designated maintenance period. Any UPDATE STATISTICS statements that do not complete are retained on the list for the next maintenance period.

Informix® Warehouse Accelerator

This includes information about installing and configuring the accelerator server, creating data marts, accelerators, and accelerated query tables (AQTs).

These topics are of interest to the following users:

- Database administrators
- System administrators

These topics are written with the assumption that you have the following background:

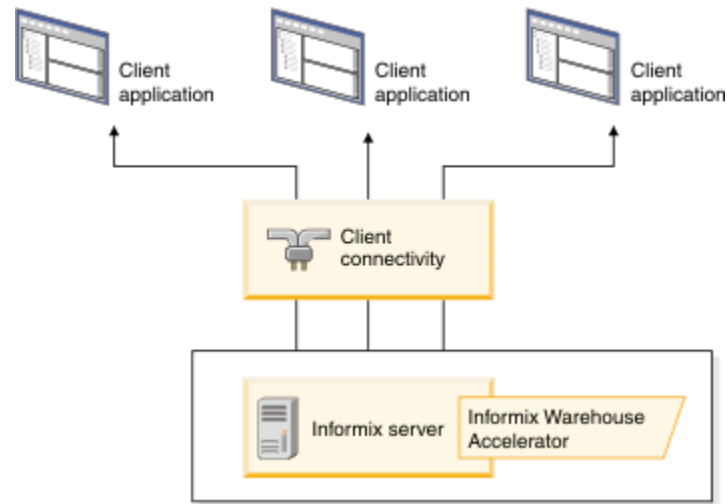
- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Experience working with relational databases or exposure to database concepts
- Experience with database server administration, operating-system administration, or network administration
- Experience with data warehousing

Overview of

is a product that you use to improve the performance of your warehouse queries.

processes warehouse queries more quickly than the Informix® database server. Improves performance while at the same time reduces administration. There are only a few configuration parameters that you need to set to tune query performance. You can install on same computer as the Informix® database server, as shown in the following figure. You can also install on a separate computer.

Figure 13. installed on the same computer as the Informix® database server.



is optimal for processing data warehouse queries. The demands of data warehouse queries are substantially different from online transactional processing (OLTP) queries. A typical data warehouse query requires the processing of a large set of data and returns a small result set.

The overhead required to accelerate a query and return a result set is negligible compared with the benefits of using :

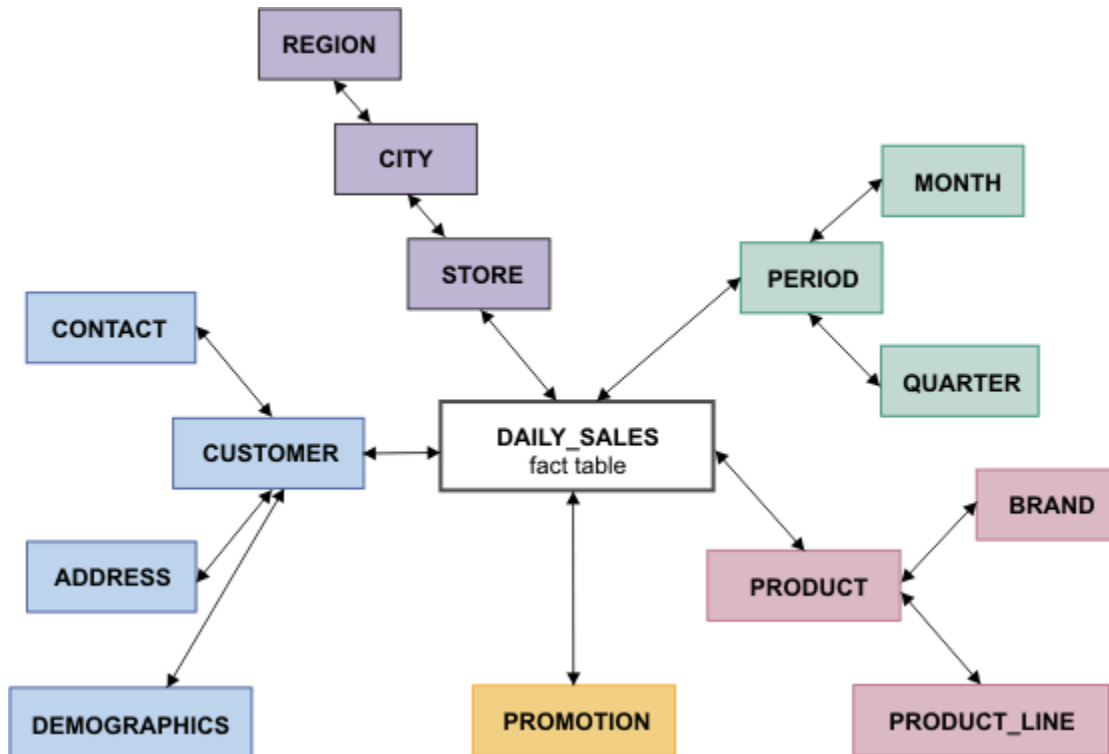
- uses a copy of the data.
- To expedite query processing, the copy of the data is kept in memory in a special compressed format. Advances in compression techniques, query processing on compressed data, and hybrid columnar organization of compressed data enable to query the compressed data.
- The data is compressed and kept in memory by to maximize parallel query processing.

In addition to the performance gain of the warehouse query itself, the resources on the database server can be better used for other types of queries, such as OLTP queries, which perform more efficiently on the database server.

You can use with a database server that supports a mixed workload OLTP database and a data warehouse database, or use it with a database server that supports only a data warehouse database. However, before you use you must design and implement a dimensional database that uses a star or snowflake schema for your data warehouse. This design includes selecting the business subject areas that you want to model, determining the granularity of the fact tables, and identifying the dimensions and hierarchies for each fact table. Additionally, you must identify the measures for the fact tables and determine the attributes for each dimension table.

The following figure shows a sample snowflake schema with a fact table and multiple dimension tables.

Figure 14. A sample snowflake schema that has the DAILY_SALES table as the fact table.



An *accelerator server* is an installation of .

The accelerator server contains one or more accelerators. An *accelerator* is a logical entity that contains information for a connection from the database server to the accelerator server and for the data marts associated with that connection. A *data mart* specifies the collection of tables from a database that are loaded into an accelerator and the relationships, or references, between these tables. After you create a data mart, information about the data mart is sent to the database server in the form of a special view called an *accelerated query table* or AQT. The database server uses the AQTs to determine which queries or query blocks can be accelerated.

Overview: Accelerating queries with Informix® Warehouse Accelerator

You can set up query acceleration with by following a few installation and configuration steps. Before you begin, plan the installation by reviewing the architecture and administration options.

1. Install, start, and prepare the Informix® database server for .

Ensure that user `informix` has write access to the `sqlhosts` file and set up a default sbspace.

2. Design and implement a dimensional database that uses a star or snowflake schema.

Ensure that the database uses the data types, functions, expressions, and joins that are supported by .

3. Install .

Determine the best installation architecture for your business needs and your hardware. You can install on the same computer as the Informix® database server, on a different computer, or on a hardware cluster. Review the installation prerequisites and requirements and determine which tool you want to use to administer .

4. Grant the appropriate permissions to users for administering . Users must be user **informix**, have the DBA role, or have the WAREHOUSE privilege.

You can run SQL routines or Java™ classes from the command line.

5. Configure and start the accelerator server.

Identify the network interface, create a storage directory, and edit the `dwainst.conf` configuration file. You then run the `ondwa setup` command to create the files and subdirectories that are required to run the accelerator server. You run the `ondwa start` command to start the accelerator server.

6. Create an accelerator by configuring a connection between the accelerator server to the database server.

You specify an accelerator name and connection information.

7. Create and load a data mart.

You can use any of the administration tools to create and load a data mart. You can design a data mart yourself or use workload analysis to design an effective data mart that is based on your data and queries. You can use workload analysis by running routines.

8. Turn on query acceleration by setting the **use_dwa** environment variable to `accelerate on`.

You run an SQL statement from the database client to set the **use_dwa** environment variable.

After you turn on query acceleration, the client application sends queries to the database server. The SQL optimizer on the database server matches the queries that can be accelerated to the AQTs and sends the queries to the accelerator server. The results are returned to the database server, which sends the results to the client application.

Related information

[Installation on page 70](#)

[Configuration on page 76](#)

[Data marts and AQTs on page 95](#)

[Tools for administering Informix Warehouse Accelerator on page 60](#)

[Permissions for administering Informix Warehouse Accelerator on page 59](#)

Informix® Warehouse Accelerator architecture options

There are several approaches that you can use to integrate into your current system architecture.

You can install the accelerator server on the same computer as your Informix® database server, on a separate computer, or on a cluster. There must be a TCP/IP connection between the database server and the accelerator server. If the accelerator server is installed on the same computer as the database server, the connection must be a local loop-back TCP/IP connection.

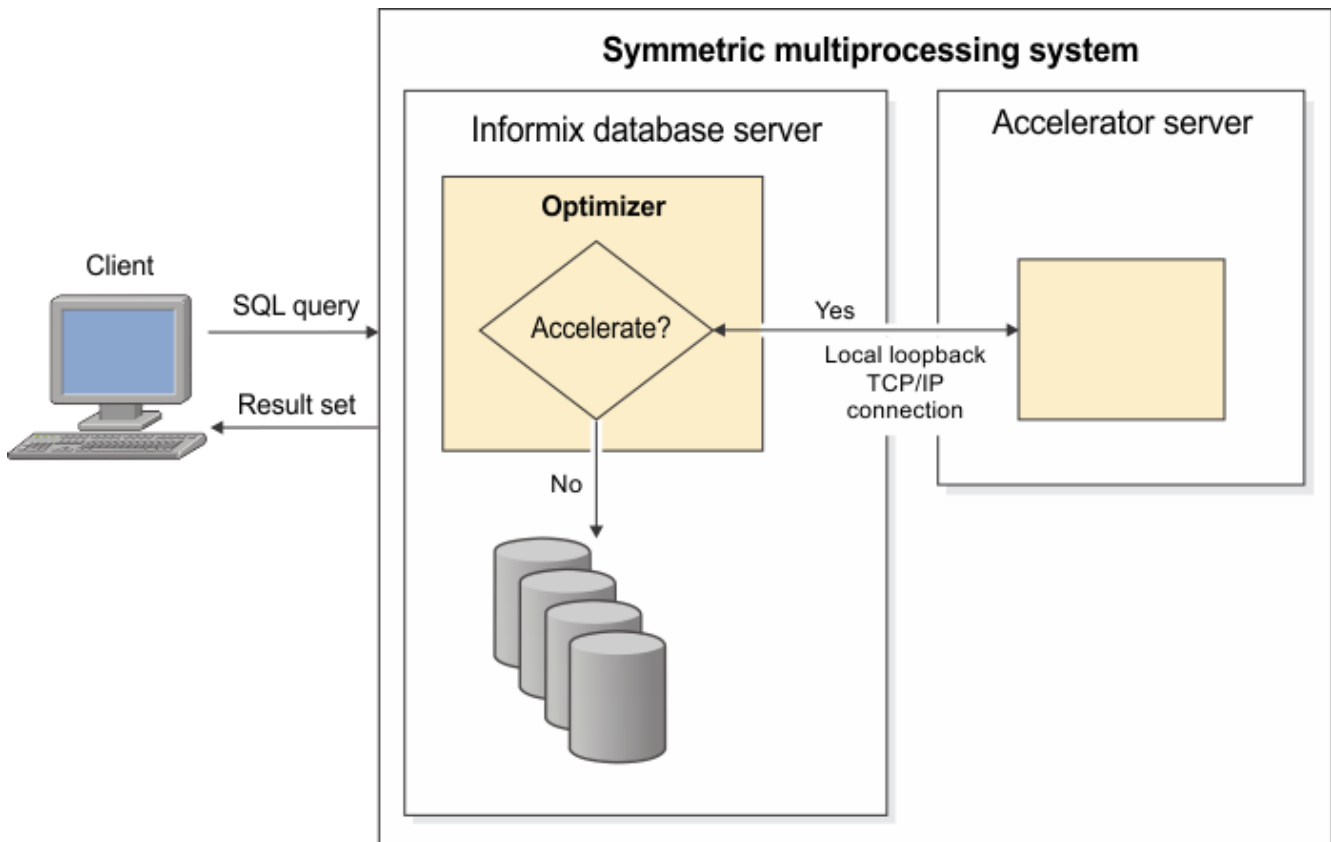
If the Informix® database server is installed in a high-availability cluster environment, you can also reduce the workload on the primary server by using one or more of the secondary servers to accelerate queries. You can also install the accelerator server on a secondary server.

The query optimizer on the database server identifies which data warehouse queries can be accelerated and sends those queries to the accelerator server. The result set is sent back to the database server, which passes the result set back to the client. If the query cannot be processed by the accelerator server, the query is processed on the database server.

The accelerator server and the database server on the same computer

The following figure shows the accelerator server and the database server installed on the same computer.

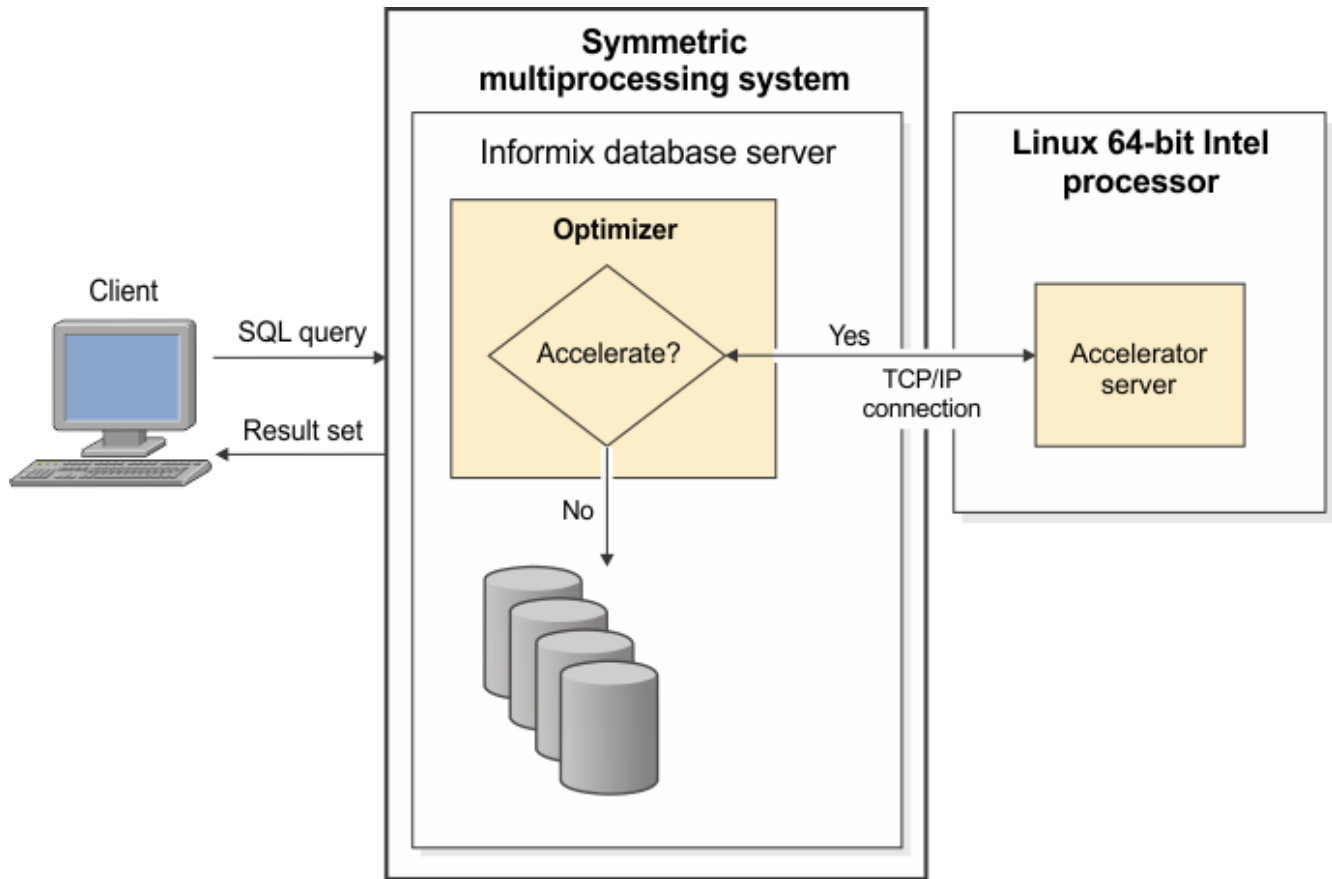
Figure 15. The accelerator server and the database server installed on the same computer.



The accelerator server and the database server on different computers

The following figure shows the accelerator server and the database server installed on different computers.

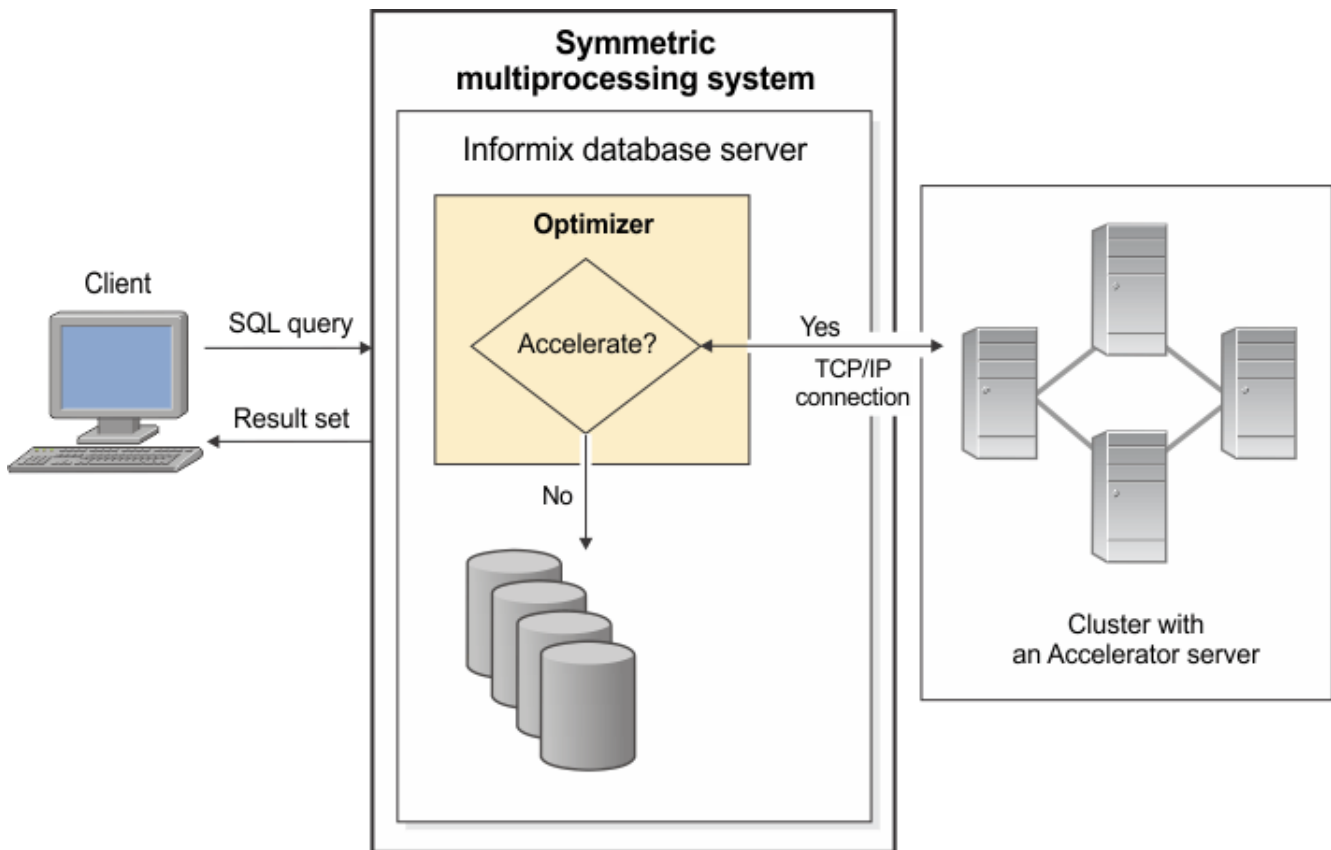
Figure 16. The accelerator server and the database server installed on different computers.



The accelerator server installed on a cluster system

The following figure shows the accelerator server installed on a cluster and the database server installed on a separate computer.

Figure 17. The accelerator server installed on a cluster system and the database server installed on a separate computer.



Informix® Warehouse Accelerator internal architecture

The accelerator server consists of two or more nodes, which are *processes* that run in the Linux™ operating system.

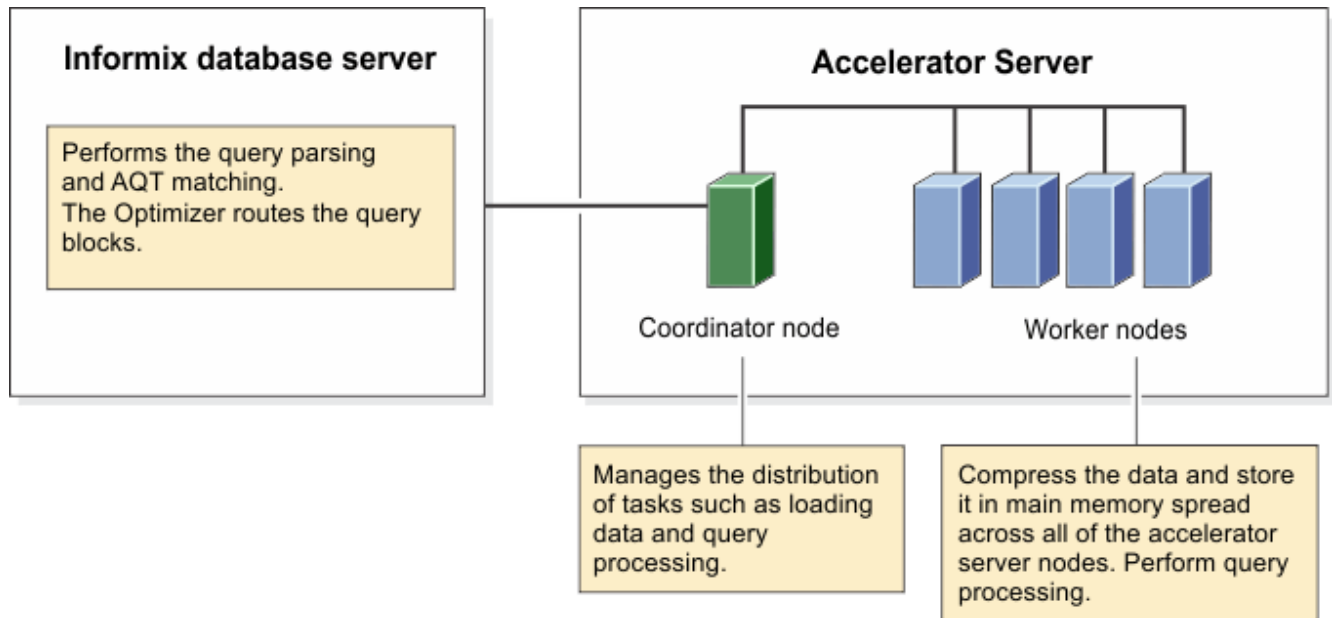
The nodes use main memory and disk space. By default, the accelerator server has one coordinator node and one worker node. The main memory is divided into shared memory for the coordinator node, shared memory for the worker node, and private memory that is used by the processes and their threads on an as-needed basis.

The coordinator node manages the requests from the database server such as loading data and running queries. The database server and the *ondwa* utility connect to the coordinator node. The coordinator node communicates with the worker node. The worker node has all of the data in main memory in a compressed format and processes the queries. You can configure more worker nodes to obtain higher parallelism for certain environments.

The processes are multithreaded so parallelization is in effect even when there is only one coordinator and one worker node. You use the *ondwa* utility to start the accelerator server nodes.

The following figure shows that the database server communicates with the worker nodes through the coordinator node, and describes the roles of the database server, the coordinator node, and the worker nodes.

Figure 18. A sample accelerator node architecture with one coordinator node and four worker nodes.



Permissions for administering Informix® Warehouse Accelerator

You must have permissions to administer .

To administer with SQL routines, Java™ classes, you must be user **informix** and have the WAREHOUSE privilege.

WAREHOUSE privilege

You grant the WAREHOUSE privilege by running the `admin()` or `task()` function with the `grant admin` and `WAREHOUSE` arguments. For example, the following statement, run in the **sysadmin** database, grants the user Bob the WAREHOUSE privilege:

```
EXECUTE FUNCTION task("grant admin", "Bob", "WAREHOUSE");
```

If the first routine is run in a new database, the `EXTEND` role, `RESOURCE` role, and language-level privileges permissions are required. When a routine is run in a new database, the initial creation of all functions and procedures is triggered. For subsequent runs these permissions are not required.

EXTEND role privilege

If `IFX_EXTEND_ROLE=1` in the `onconfig` file, you must grant the `EXTEND` role privilege to all users besides **informix** and the `DBA` role, that perform functions and procedures in your database. For example, this statement grants the `EXTEND` role to user Bob:

```
GRANT EXTEND TO 'Bob';
```

RESOURCE role and language-level privileges

If `IFX_EXTEND_ROLE=0` in the `onconfig` file, you must grant the RESOURCE role and language-level privileges to all users besides **informix** that perform functions and procedures in your database. For example, these statements grant the RESOURCE role and language-level privileges to user Bob:

```
GRANT RESOURCE TO 'Bob';  
GRANT USAGE ON LANGUAGE C TO 'Bob';
```

Additional routine-specific permissions might be required and are documented in the individual routines.

Related information

[Overview: Accelerating queries with Informix Warehouse Accelerator on page 54](#)

[SQL administration routines on page 134](#)

[Java classes for Informix Warehouse Accelerator on page 60](#)

[Tools for administering Informix Warehouse Accelerator on page 60](#)

Tools for administering Informix® Warehouse Accelerator

You can use these tools to administer .

You must have permissions to run the administration tools.

All the tools work through a database connection from the database server to the accelerator server. They do not interact directly with the accelerator server.

- SQL administration routines. You can use these functions and procedures from any SQL client from the command line or in scripts to automate tasks.
- Java™ classes are included in the software. You can use the Java™ classes from the command line or in scripts so that you can automate tasks.

Related information

[Java classes for Informix Warehouse Accelerator on page 60](#)

[SQL administration routines on page 134](#)

[Permissions for administering Informix Warehouse Accelerator on page 59](#)

Java™ classes for Informix® Warehouse Accelerator

includes a set of Java™ classes that you can use from the command line or in an application.

You can use the Java™ classes to create and drop data marts, update connectivity information between the Informix® database server and the accelerator server, and more. You can use the classes to automate the steps for refreshing the data

stored in the data marts. Instead of dropping and re-creating the data marts manually, you can create an application that runs when it is convenient for your organization.

You must have permissions to run the Java™ classes.

The files for the Java™ classes are in this directory: `dwa/example/cli`. The directory includes the following files:

conn.prop.std

Template for the connection properties file.

dwa_java_quickref.txt

Lists the requirements and setup steps with examples. Also lists the classes and usage.

dwa_java_reference.txt

Provides the full syntax for all the classes. Includes recommendations and example XML schemas.

dwa_message_reference.txt

Lists the messages in numerical order. These messages are not exclusive to the Java™ classes. They might be displayed when you use other administration tools for .

dwaccli.jar

Contains the executable code for the Java™ classes.

env.std

Template for setting the CLASSPATH environment variable.

Related information

[Permissions for administering Informix Warehouse Accelerator on page 59](#)

Accelerated queries

If a query or a query block matches an AQT (accelerated query table), it is sent to the accelerator server. This process is called *acceleration*. The results are then returned from the accelerator server to the Informix® database server. Different types of queries are more or less suitable for acceleration.

Queries that benefit from acceleration

The database server uses information about the existing data marts to determine which types of queries are sent to the accelerator server for processing.

The following types of queries benefit the most from being sent to the accelerator server:

- Complex, ad hoc queries that look for trends or exceptions
- Queries that access a large subset of the database, often by using sequential scans
- Queries that involve aggregation functions such as COUNT, SUM, AVG, MAX, MIN, and VARIANCE

- Queries that often create reports that group data by time, product, geography, customer set, or market
- Queries that involve star joins or snowflake joins of a large fact table with several dimension tables

Only SELECT queries that refer to a fact table, or SELECT queries that join a fact table with one or more dimension tables are processed by the accelerator server.

The same restrictions apply when a query has a query block that contains an online analytical processing (OLAP) window function or UNION, or UNION ALL set operators.

- When a query contains an OLAP window function, the database server uses for the underlying SELECT, JOIN, GROUP BY, and PROJECT operations.
- When a query contains a UNION or UNION ALL set operator, evaluates each SELECT statement separately. If either SELECT statement is eligible for query acceleration, it is sent to for processing.

The characteristics of a query or a query block determine if it can be processed by the accelerator server. For a query to be accelerated it must match the following conditions:

- The query must reference only the tables and columns that are included in the data mart definition.
- The query must use only the supported data types.
- The query must reference a table that is marked as a fact table in the data mart definition.
- The query must use only the supported join types.
- The query must use only supported aggregate and scalar functions.

Related reference

[Supported and unsupported joins on page 69](#)

[Supported functions and expressions on page 67](#)

Related information

[Types of queries that are not accelerated on page 65](#)

[Accelerated query tables on page 95](#)

Query Example: Quantity, revenue, and cost by item

This example shows, for a given category, all of the items including quantity sold, revenue, and cost.

Example

```
SELECT ITEM_DESC,
       SUM(QUANTITY_SOLD),
       SUM(EXTENDED_PRICE),
       SUM(EXTENDED_COST)
FROM PERIOD, DAILY_SALES, PRODUCT, STORE, PROMOTION
WHERE PERIOD.PERKEY=DAILY_SALES.PERKEY AND
       PRODUCT.PRODKEY=DAILY_SALES.PRODKEY AND
```



```

STORE.STOREKEY=DAILY_SALES.STOREKEY AND
PROMOTION.PROMOKEY=DAILY_SALES.PROMOKEY AND
CALENDAR_DATE BETWEEN '2011/01/01' AND '2011/01/31' AND
STORE_NUMBER=01 AND
PROMODESC IN ('Advertisement', 'Coupon', 'Weekly Special',
              'Overstocked Items') AND
CATEGORY=42
GROUP BY ITEM_DESC;

```

Query Example: Profit by store

This example shows the profit by store for a given category on a given day.

Example

```

SELECT T1.STORE_NUMBER,T1.CITY,T1.DISTRICT,SUM(AMOUNT) AS SUM_PROFIT
FROM
  (SELECT STORE_NUMBER,STORE.CITY,DISTRICT, EXTENDED_PRICE-EXTENDED_COST
   FROM PERIOD, PRODUCT,STORE, DAILY_SALES
   WHERE
     PERIOD.CALENDAR_DATE=3/1/2011 AND
     PERIOD.PERKEY=DAILY_SALES.PERKEY AND
     PRODUCT.PRODKEY=DAILY_SALES.PRODKEY AND
     STORE.STOREKEY=DAILY_SALES.STOREKEY AND
     PRODUCT.CATEGORY=42 ) AS T1(STORE_NUMBER,CITY,DISTRICT,AMOUNT)
GROUP BY DISTRICT,CITY, STORE_NUMBER
ORDER BY DISTRICT,CITY, STORE_NUMBER DESC;

```

Query Example: Revenue by store for each brand

This example takes the revenue for each brand and calculates the revenue by store.

Example

Products are grouped by store, current week, prior week, and prior month totals.

```

SELECT STORE_NUMBER,
  SUM(CASE WHEN ((CALENDAR_DATE >= 8/8/2010)
                AND (CALENDAR_DATE < 8/14/2010))
        THEN EXTENDED_PRICE ELSE 0 END) AS CURR_PERIOD,
  SUM(CASE WHEN ((CALENDAR_DATE >= 8/1/2010)
                AND (CALENDAR_DATE <= 8/7/2010))
        THEN EXTENDED_PRICE ELSE 0 END) AS PRIOR_WEEK,
  SUM(CASE WHEN ((CALENDAR_DATE >= 7/1/2010)
                AND (CALENDAR_DATE <= 7/28/2010))
        THEN EXTENDED_PRICE ELSE 0 END) AS PRIOR_MONTH
FROM PERIOD,PRODUCT,DAILY_SALES,STORE
WHERE PRODUCT.PRODKEY=DAILY_SALES.PRODKEY
      AND PERIOD.PERKEY=DAILY_SALES.PERKEY
      AND STORE.STOREKEY=DAILY_SALES.STOREKEY
      AND CALENDAR_DATE BETWEEN 7/1/2010 and 8/14/2010
      AND ITEM_DESC LIKE 'NESTLE%'
GROUP BY STORE_NUMBER
ORDER BY STORE_NUMBER;

```

Query Example: Week to day profits

This example shows the week to day profits for a given category within region.

Example

```
SELECT FIRST 100 SUB_CATEGORY_DESC,
       SUM(CASE REGION WHEN 'North'
            THEN EXTENDED_PRICE-EXTENDED_COST ELSE 0
            END) AS NORTHERN_REGION,
       SUM(CASE REGION WHEN 'South'
            THEN EXTENDED_PRICE-EXTENDED_COST ELSE 0
            END) AS SOUTHERN_REGION,
       SUM(CASE REGION WHEN 'East'
            THEN EXTENDED_PRICE-EXTENDED_COST ELSE 0
            END) AS EASTERN_REGION,
       SUM(CASE REGION WHEN 'West'
            THEN EXTENDED_PRICE-EXTENDED_COST ELSE 0
            END) AS WESTERN_REGION,
       SUM(CASE WHEN REGION IN ('North', 'South', 'East', 'West')
            THEN EXTENDED_PRICE-EXTENDED_COST ELSE 0
            END) as ALL_REGIONS
FROM PERIOD per, PRODUCT prd, STORE st, DAILY_SALES s
WHERE
  per.PERKEY=s.PERKEY AND
  prd.PRODKEY=s.PRODKEY AND
  st.STOREKEY=s.STOREKEY AND
  per.CALENDAR_DATE BETWEEN '07/01/2010' AND '09/30/2010' AND
  CATEGORY<>88
GROUP BY SUB_CATEGORY_DESC
ORDER BY SUB_CATEGORY_DESC;
```

Query block example: Average employee sales

This example of a query uses OLAP window aggregation functions.

In this example, processes the join between the `employee` and `sales` tables, and then returns the result set to the Informix® database server. The Informix® database server then processes the OLAP functions, RANK and AVG.

```
SELECT e.emp_name,
       RANK() OVER (PARTITION BY region
                   ORDER BY total_sales desc),
       AVG(sales) OVER (PARTITION BY region, year)
FROM employee e, sales s
WHERE e.emp_id = s.emp_id;
```

Related reference

[Supported functions and expressions on page 67](#)

Query block example: Sales count

In this query, the query block `SELECT COUNT(*) FROM sales` is sent to for acceleration.

The query block `SELECT COUNT(*) FROM systables` does not qualify for acceleration.

```
SELECT COUNT(*) FROM sales UNION ALL SELECT COUNT(*) FROM systables;
```

In this example, both SELECT query blocks are sent to for processing. Each query block is sent separately.

```
SELECT COUNT(*) FROM sales UNION ALL SELECT COUNT(*) FROM sales_returns;
```

Related information

[Types of queries that are not accelerated on page 65](#)

Types of queries that are not accelerated

There are characteristics of queries that either will not benefit from being sent to the accelerator server, or will not be considered for acceleration.

Queries that will not benefit from being accelerated

Queries that only refer to a single, small dimension table do not benefit from being sent to the accelerator server for processing as much as queries that also refer to a fact table.

Queries that return a large result set should be processed by the database server to avoid the overhead of sending the large result set from the accelerator server over the connection to the database server. If a query returns millions of rows, the total response time of the query is influenced by the maximum transfer rate of the connection. For example, the following query might return a very large result set:

```
SELECT * FROM fact_table ORDER BY sales_date;
```

Queries that search only a small number of rows of data should be processed by the database server to avoid the overhead of sending the query to the accelerator server.

Queries that are not considered for acceleration

There are some queries that will not be processed by the accelerator server.

Queries that would change the data cannot be processed by the accelerator server and must be processed by the database server. The data in the accelerator server is a snapshot view of the data and is read only. There is no mechanism to change the data in the data marts and replicate those changes back to the source database server.

Other queries that are not processed by the accelerator server include queries that contain INSERT, UPDATE, or DELETE statements, queries that contain subqueries, and other OLTP queries.

Related information

[Queries that benefit from acceleration on page 61](#)

Supported data types

supports specific data types.

The following data types are supported:

- BIGINT
- BIGSERIAL
- CHAR
- CHARACTER
- CHARACTER VARYING
- DATE
- DATETIME HOUR TO HOUR
- DATETIME HOUR TO MINUTE
- DATETIME HOUR TO SECOND
- DATETIME YEAR TO DAY
- DATETIME YEAR TO FRACTION(x)
- DATETIME YEAR TO HOUR
- DATETIME YEAR TO MINUTE
- DATETIME YEAR TO MONTH
- DATETIME YEAR TO SECOND
- DATETIME YEAR TO YEAR
- DECIMAL(p,s), with $p \leq 31$.
- DOUBLE PRECISION
- FLOAT
- INT
- INT8
- INTEGER
- LVARCHAR
- MONEY
- NUMERIC(p,s), with $p \leq 31$.
- REAL
- SERIAL
- SERIAL8
- SMALLFLOAT
- SMALLINT
- VARCHAR

Restrictions

- Arithmetic operations that include values of different DATE or DATETIME type variants that are not supported by are automatically run by Informix®, unless the fallback option is set to off. For example, Informix® supports subtracting a DATETIME YEAR TO MINUTE value from a DATETIME YEAR TO MONTH value, but does not.

Supported functions and expressions

Queries with supported functions and expressions are accelerated on .

Aggregate functions and expressions

The following aggregate functions are supported by :

- AVG
- COUNT
- STDEV
- SUM
- VARIANCE

Aggregate functions with the DISTINCT keyword are supported. For example, COUNT DISTINCT(). The DISTINCT keyword is also supported in an SQL projection clause like `SELECT DISTINCT stock_num, manu_code FROM items;`. However, you cannot use the DISTINCT keyword in an aggregate function and in an SQL projection clause in the same query.

OLAP window functions

The following OLAP functions are supported by Informix®. This list includes several aggregate functions that can be specified as OLAP window aggregates when the OVER clause immediately follows the function, or as ordinary aggregates that operate on the entire result set, if no OVER clause is specified:

- AVG
- COUNT
- CUME_DIST
- DENSE_RANK or DENSERANK
- FIRST_VALUE
- LAG
- LAST_VALUE
- LEAD
- MAX
- MIN
- NTILE
- PERCENT_RANK
- RANGE
- RANK
- RATIO_TO_REPORT or RATIOTOREPORT
- ROW_NUMBER or ROWNUMBER
- STDEV
- SUM
- VARIANCE

User-defined functions

User-defined functions are not supported.

Scalar functions

The following scalar functions are supported by :

- ABS
- ADD_MONTHS
- CASE
- CEIL
- CONCAT
- CURRENT
- DATE
- DAY
- DECODE
- FLOOR
- LAST_DAY
- LEN
- LENGTH
- LOWER
- LPAD
- LTRIM
- MAX
- MIN
- MOD
- MONTH
- MONTHS_BETWEEN
- NEXT_DAY
- NVL
- POW
- POWER
- QUARTER
- RANGE
- ROUND
- RPAD
- RTRIM
- SQRT
- SUBSTR
- SUBSTRING
- SYSDATE
- TODAY
- TRIM

- TRUNC
- UNITS DAY
- UNITS FRACTION
- UNITS HOUR
- UNITS MINUTE
- UNITS MONTH
- UNITS SECOND
- UNITS YEAR
- UPPER
- WEEKDAY
- YEAR

Restrictions

- The formatting rules that are specified when you connect to the Informix® server are not necessarily applied by when converted to character data types. Here are examples of how implicit and explicit casts to character data types are affected:
 - If a query specifies an explicit cast of a DATE or DATETIME value to a character type, for example `... date_column::char(10) ...`, the date value is formatted by as `yyyy-mm-dd`. If this cast appears in the projection list, then the query result is displayed in this format. If it appears as a predicate, the predicate produces a match only if the counterpart is formatted in the same way.
 - If a query specifies an implicit cast to a character data type, for example, by using the CONCAT function or the equivalent concatenation operator (||), formats the output according to the default Informix® server rules.
- Support of the CURRENT and SYSDATE functions is limited to the same variants that are supported for the DATETIME data type, as documented in [Supported data types on page 65](#).
- Queries that use the ROUND function or the TRUNC function with the DATE data type fail when the query is sent to the for processing. An error message about DRDA® AR PREPARE is returned.
- The SUBSTR and the SUBSTRING functions have the following restrictions in :
 - The *start_position* value must be greater than 0.
 - The *length* value must be greater than or equal to 0.
 - The sum of the *start_position* value and the *length* value must be less than or equal to the length of the *source_string*.

If the parameter values are outside of these boundaries, the accelerated query will cause an error message.

Related information

[Queries that benefit from acceleration on page 61](#)

Supported and unsupported joins

Specific join types, join predicates, and join combinations are supported by .

Supported joins

Equality join predicates, INNER joins, and LEFT OUTER joins are the supported join types.

The fact table referenced in the query must be on the left side of the LEFT OUTER join.

Any filter on a dimension table that you use in a LEFT OUTER join must be a post-join filter. A post-join filter is in the WHERE clause after the join.

Unsupported joins

The following joins are not supported:

- RIGHT OUTER joins
- FULL OUTER joins
- Informix® outer joins
- Joins that do not use an equality predicate
- Subqueries

Related information

[Queries that benefit from acceleration on page 61](#)

Installation

You can install on the same computer as the Informix® database server, on a separate computer, or on a cluster (including a high-availability cluster).



Important: Only one instance of can be installed on a computer.

Before you install the , ensure that your computers meet the software and hardware prerequisites, and that you have decided which architecture you want to implement.

prerequisites and requirements

Ensure that your system meets the prerequisites and the requirements for . You must have certain utilities installed. Cluster installations have additional requirements.

General prerequisites

Following are the general prerequisites for :

- Only one instance of can be installed on a computer.
- Both the Informix® and versions must be at the same release and fix pack level.

System requirements

must be installed on a computer that uses a Linux™ Intel™ x86 64-bit operating system. For the detailed list of supported operating systems and hardware, see the system requirements.

Required libraries

You must have the following libraries installed on the server where is installed:

- libicu
- xerces-c

Required utilities

You must have the following utilities on the server where is installed:

- Telnet client program
- Expect utility (expect-5)
- su command

Requirements for installed on a cluster system

Following are requirements for installing on a cluster system:

- You must have a shared-disk cluster file system. For example, General Parallel File System (GPFS™).
- For maximum performance in a shared cluster environment, all cluster nodes must have the same amount of memory and processor cores.
- The user root must be able to connect to all cluster nodes by using the Secure Shell (SSH) network protocol without a password.
- If user **informix** is used for administration, user **informix** must be able to connect to all cluster nodes by using the Secure Shell (SSH) network protocol without a password.
- The file paths for the installation files must be the same on each cluster node.

Related information

[Configuring memory for Informix Warehouse Accelerator on page 83](#)

Informix® Warehouse Accelerator directory structure

When you install and set up , there are several directories that are needed.

Installation directory

is installed in the directory that is specified by the INFORMIXDIR environment variable, if the variable is set in the environment in which the installer is launched. If the variable is not set, the default installation directory is `/opt/IBM/informix`.

Whenever there is a reference to the file path for the *accelerator server installation directory*, the file path appears as `$IWA_INSTALL_DIR`.

Storage directory

The software resides in its own directory, referred to as the *accelerator server storage directory*. This directory stores the catalog, data marts, logs, traces, and so forth. You create this directory when you configure the accelerator server. The file path for this directory is stored in the DWADIR parameter in the `dwainst.conf` file.

Sample directory for Java™ classes

The Java™ classes that are included with the command line sample are located in the `dwa/example/cli` directory.



Tip: Information about the Java™ classes is located in the `dwa_java_reference.txt` file in the `dwa/example/cli` directory.

Related reference

[dwainst.conf configuration file on page 80](#)

Related information

[Configuring Informix Warehouse Accelerator on page 76](#)

Preparing the Informix® database server

Before you install , configure the database server.

Before you begin

The **sysadmin** database must exist in the Informix® database server.

To configure the Informix® database server:

1. Ensure that the user **informix** has write access to the `sqlhosts` file and the directory that the file is in.
2. Define a SOCTCP network connection type in the `sqlhosts` file for the connection between the accelerator server and the database server.
3. If you do not already have a default sbspace created and configured, create the default sbspace:
 - a. In the `onconfig` file, set the SBSPACENAME configuration parameter to the name of your default sbspace.

Example

For example, to name the default sbspace `sbspl`:

```
SBSPACENAME sbspace1 # Default sbspace name
```

You must update the `onconfig` file before you start the database server.

- b. Use the `onspaces` command to create the sbspace.

Example

The following example creates an sbspace named `sbspace1`:

```
onspaces -c -S sbspace1 -p $INFORMIXDIR/tmp/sbspace1 -o 0 -s 30000
```



Note: The size of the sbospace can be relatively small, for example 30 - 50 MB.

c. Restart the Informix® database server.

4. **Optional:** Add a dwavp virtual processor. If not explicitly added, a single dwavp virtual processor is dynamically allocated when the first related activity occurs.

Choose from:

- To add a single dwavp virtual processor for the database server instance: `onmode -p +1 dwavp`.
- To permanently add a single dwavp virtual processor, add the following entry to the VPCLASS parameter in the `onconfig` file: `VPCLASS dwavp,num=1`.



Tip: In systems with significant data mart administration activity, you can define two dwavp virtual processors to avoid the delay of other administrative commands while loading data marts. For example, to add two dwavp virtual processors for the database server instance: `VPCLASS dwavp,num=2`.

Related information

[Missing sbospace on page 170](#)

Verifying the Informix® database server environment

You can use the `ondwachk` script to confirm that the Informix® database server environment is set up correctly for and that the database server can connect to the accelerator server.

Before you begin



Prerequisites:


- You must be logged on as user `informix`.
- The environment of the Informix® server instance is set.

About this task

The `ondwachk` script is part of the Informix® server installation and is located in the `$INFORMIXDIR/bin` directory.

The `ondwachk` script verifies the following Informix® server instance prerequisites for :

- The Informix® and accelerator versions match
- The sbospace is added to Informix®
- The SBSPACENAME configuration parameter is configured correctly in the **onconfig** file
- The listen thread for SOCTCP is running
- The dwavp virtual processor is running on the standard or primary server if the scheduler is stopped

- There is a writable `sqlhosts` file and directory
- An entry for an accelerator is included in the `sqlhosts` file
-  **For secondary servers only:**
 - The dwavp virtual processor is running
 - The secondary server can be updated

To verify the Informix® database server environment:

Run the `ondwachk` script.

Installing

You can use the graphical mode, console mode, or silent mode to install .

Before you begin

Before you begin, see the [prerequisites and requirements on page 70](#).

About this task

You can install on the same computer as your Informix® database server, on a separate computer, or on a cluster system.

You can install from the provided installation media, or you can install it after you download Informix®.

1. On the computer where you want to run the installation program, log in as user `root`.
2. From the product media or the download site, locate the HCL Informix® bundle and unpack the `iif.version.tar` file.
3. Select the installation mode that you want to use:

Choose from:

- For the graphical or console mode:
 - a. Issue the `install` command to start the installation program:

Installation mode	Installation command
Graphical	<code>./iwa_install -i gui</code>
Console	<code>./iwa_install -i console</code>

- b. Read the license agreement and accept the terms.
 - c. Respond to the prompts in the installation program as the program guides you through the installation.
- For the silent mode:
 - a. Make a copy of the response file template that is located in the same directory as the installation program. The name of the template is `iwa.properties`.
 - b. In the response file change the value for `license` from `FALSE` to `TRUE`, to indicate that you accept the license terms. For example:

```
DLICENSE_ACCEPTED=TRUE
```

c. Issue the installation command for the silent mode. The command for the silent mode installation is:

```
./iwa_install -i silent -f "file_path"
```



Tip: Specify the absolute path for the response file.

For example, to use the silent mode with a response file called `installer.properties` that is located in the `/usr3/iwa/` directory, the command is:

```
./iwa_install -i silent -f "/usr3/iwa/installer.properties"
```

Results

- is installed in the directory that is specified by the `INFORMIXDIR` environment variable, if the `INFORMIXDIR` environment variable is set in the environment in which the installer is launched. Otherwise, the default installation directory is `/opt/IBM/informix`.
- The configuration file, `$IWA_INSTALL_DIR/dwa/etc/dwainst.conf`, is generated during the installation. This configuration file is required to start .
- The installation log file, `$IWA_INSTALL_DIR/IBM/Informix_Warehouse_Accelerator_InstallLog.log`, is generated during the installation. This log file provides information on the actions performed during installation and success or failure status of those actions.

What to do next

You must configure and start before you can use it.

Related information

[Configuration on page 76](#)

[Uninstalling Informix Warehouse Accelerator on page 75](#)

[Configuring memory for Informix Warehouse Accelerator on page 83](#)

Uninstalling Informix® Warehouse Accelerator

If you need to reinstall the or if you no longer want to use , you must uninstall it.

Before you begin

You must be logged on as user root to run the uninstaller.

About this task

To uninstall :

1. Stop the accelerator server by using the `ondwa stop` command.
2. **Optional:** To completely uninstall the files, for example in cases where you do not plan to reinstall or upgrade the product, remove the files in the shared and local directories in the accelerator server storage directory and the log files for the accelerator server node that were created by the `ondwa setup` command:
 - a. Drop all data marts from all accelerators of the accelerator server.
 - b. Remove all accelerators of the accelerator server.
 - c. Run the `ondwa clean` command.
3. Run the uninstaller program, `uninstall_iwa`, which is located in the `$INFORMIXDIR/uninstall/uninstall_iwa` directory.

Related reference

[ondwa stop command on page 93](#)

[ondwa clean command on page 94](#)

Related information

[Installing on page 74](#)

[Drop a data mart on page 118](#)

Configuration

You must configure the accelerator server to work with the Informix® database server.

After you install , there are several configuration steps that you must complete before it can accelerate queries.

Related information

[Installing on page 74](#)

Configuring Informix® Warehouse Accelerator

You must configure the before you can enable query acceleration and set up the connection between the accelerator server and the database server.


About this task

Configure the accelerator server by identifying the network interface, creating a storage directory, and editing the `dwainst.conf` configuration file:


1. On the computer where the accelerator server is installed, log on as user root.
2. Determine the correct network interface value to use for the connection from the Informix® database server to the accelerator server:

- a. Run the Linux™ `ifconfig` system command to retrieve the information about the network devices on your system.
- b. Review the output with your system administrator and network administrator and select the appropriate value to use.

Examples of network interface values are: `eth0`, `lo`, `peth0`. The default value is `lo`.

 **Important:** If the accelerator server is installed on a separate computer than the Informix® database server, you cannot use the local loopback value.

3. Create a directory to use as the *accelerator server storage directory*. Create this directory with enough space to store the accelerator server catalog, data marts, logs, traces, and so on.
Because the amount of data in the accelerator server storage directory might increase significantly, do not create the accelerator server storage directory in the accelerator server installation directory. You will specify the file path for this directory in the value for the `DWADIR` parameter in the `dwainst.conf` file.
4. Open the `$IWA_INSTALL_DIR/dwa/etc/dwainst.conf` configuration file. Review and edit the values in the [dwainst.conf configuration file on page 80](#).

 **Important:** Specify the network interface value for the `DRDA_INTERFACE` parameter in the `dwainst.conf` file.

5. Run the [ondwa setup command on page 88](#) to create the files and subdirectories that are required to run the accelerator server.
6. Run the [ondwa start command on page 89](#) to start all of the accelerator server nodes.

Related reference

- [The ondwa utility on page 86](#)
- [ondwa setup command on page 88](#)
- [ondwa start command on page 89](#)
- [dwainst.conf configuration file on page 80](#)

Related information

- [Informix Warehouse Accelerator directory structure on page 71](#)

Configuring Informix® Warehouse Accelerator for hardware clusters

You must configure before you can enable query acceleration and set up the connection between the accelerator server and the database server. In a cluster system, one accelerator server coordinator node uses the first cluster node, and then each cluster node that you add to the cluster becomes an accelerator server worker node.

Before you begin

Prerequisite: Test that user root or user informix can run the Secure Shell (SSH) network protocol without a password between all cluster nodes.

About this task

Configure the accelerator server by identifying the network interface, creating a storage directory, editing the parameters in the `dwainst.conf` configuration file, and creating a `cluster.conf` file.

1. On one of the cluster nodes where the accelerator server is installed, log on as user root.
2. Determine the correct network interface value to use for the connection from the Informix® database server to the accelerator server:
 - a. Run the Linux™ `ifconfig` system command to retrieve the information about the network devices on your system.
 - b. Review the output with your system administrator and network administrator and select the appropriate value to use.

Examples of network interface values are `eth0` or `peth0`.

3. On the shared cluster file system, create a directory to use as the *accelerator server storage directory*. The storage directory must be accessible with the same path on all cluster nodes. Create this directory with enough space to store the accelerator server catalog, data marts, logs, traces, and so on.

For example:

```
$ mkdir $IWA_INSTALL_DIR/dwa/demo
```

Because the amount of data in the accelerator server storage directory might increase significantly, do not create the accelerator server storage directory in the accelerator server installation directory.

4. Open the `$IWA_INSTALL_DIR/dwa/etc/dwainst.conf` configuration file. Review and edit the values in the [dwainst.conf configuration file on page 80](#):
 - a. For the `DRDA_INTERFACE` parameter, specify the network interface value that you identified in [step 2 on page 78](#).
 - b. For the `DWADIR` parameter, specify the file path for the storage directory that you created in [step 3 on page 78](#).
On all cluster nodes, the `DWADIR` parameter must be the same file path.
 - c. For the `CLUSTER_INTERFACE` parameter, specify the network device name for the connection between the cluster nodes.
For example, `eth0`.
 - d. If only one coordinator node or one worker node will run on each cluster node, add the following additional parameters and values:

```
CORES_FOR_SCAN_THREADS_PERCENTAGE=100
CORES_FOR_LOAD_THREADS_PERCENTAGE=100
CORES_FOR_REORG_THREADS_PERCENTAGE=25
```


5. In the `$IWA_INSTALL_DIR/dwa/etc` directory, create a file named `cluster.conf` to store the host names or IP addresses of the cluster nodes. In the `cluster.conf` file, enter one cluster node per line. For example:

```
node0001
node0002
node0003
node0004
```

The order that you list the hosts names (or their IP addresses) of the cluster nodes is the order that the cluster nodes are started or stopped with the `ondwa start` and `ondwa stop` commands.

6. Use the `ondwa` commands to set up and start the accelerator server. You can run the `ondwa` commands from any node in the cluster. The `ondwa` commands apply to all the nodes listed in the `cluster.conf` file.
- a. Run the [ondwa setup command on page 88](#) to create the files and subdirectories that are required to run the accelerator server.

Example output:

```
Checking for DWA_CM_node0 on node0001: stopped
Checking for DWA_CM_node1 on node0002: stopped
Checking for DWA_CM_node2 on node0003: stopped
Checking for DWA_CM_node3 on node0004: stopped
```

- b. Run the [ondwa start command on page 89](#) to start all of the cluster nodes.

Example output:

```
Starting DWA_CM_node0 on node0001: started
Starting DWA_CM_node1 on node0002: started
Starting DWA_CM_node2 on node0003: started
Starting DWA_CM_node3 on node0004: started
```

Related reference

[dwainst.conf configuration file on page 80](#)

Configuring on secondary servers in a high-availability environment

You can create data marts, load data mart data, and accelerate queries on one or more secondary servers in a high-availability cluster. By using secondary servers for , you have greater flexibility in a mixed-workload environment. For example, you can dedicate the primary server for OLTP transactions and use a secondary server for processing the warehouse analytic queries. Configure the secondary servers by modifying the `ONCONFIG` file of each secondary server, connecting one of the secondary servers to an accelerator, and copying the `sqlhosts` entry in the `sqlhosts` file to each server in the cluster.

Before you begin

The following prerequisites are required:

- Configure the high-availability cluster for the Informix® database server.

About this task

High-availability secondary servers include shared-disk (SD) secondary servers, high-availability data replication (HDR) secondary servers, and remote stand-alone (RS) secondary servers.

To configure the servers in a high-availability cluster to use , follow these steps. You can choose to configure on a subset of the secondary servers in the cluster.

1. Set the UPDATABLE_SECONDARY configuration parameter in the ONCONFIG file on each secondary server that you plan to use for .
This configuration parameter sets the number of connections to establish to the primary server. For the connection, you can set this value to 1. (Depending on the number and workload of your client applications, you might need to set this configuration parameter to a higher value.)
2. Add a dwavp virtual processor to the VPCLASS configuration parameter in the ONCONFIG file on each secondary server that you plan to use for .

Example

For example, `VPCLASS dwavp,num=1`.

For more information, see [Preparing the Informix database server on page 72](#).

3. Make sure that each secondary server has access to the required sbspace as described in [Preparing the Informix database server on page 72](#).
4. Restart each secondary server for which you modified the ONCONFIG file for the changes to take effect.
5. Connect one of the servers in the cluster to an accelerator. See [Create an accelerator on page 94](#).
When the accelerator is connected to the Informix® database server, the `sqlhosts` file on the Informix® database server is updated. You will use the new `sqlhosts` entry in the `sqlhosts` file in the next step.
6. Copy the `sqlhosts` entry that you created in the previous step to the `sqlhosts` file in each server in the cluster that you plan to use for .



Important: Do not change any of the text in the `sqlhosts` entry. The `sqlhosts` entry is stored in the server system catalog and the `sqlhosts` entry must be identical on all the servers so that they can communicate with .

Results

After you configure the servers in a high-availability cluster for , when you deploy a data mart from the primary server or any of the secondary servers, the data mart definitions are replicated to all the servers in the cluster. Any changes to the state of a data mart that occur on one of the servers are replicated to all the servers in the cluster.



Important: If the primary server fails while you are performing a data mart operation (for example, creating or loading a data mart), the data mart operation might fail. After a secondary server successfully becomes a new primary server, try any failed data mart operations again. If the primary failover caused the deploying or the loading of the data mart to fail, you must drop the data mart, create it, and load it again.

dwainst.conf configuration file

The `dwainst.conf` configuration file contains the parameters that are used to configure the accelerator server.

The `dwainst.conf` configuration file is added to the `$IWA_INSTALL_DIR/dwa/etc` directory when you install .

Open the `dwainst.conf` configuration file and edit the parameter values before you run the `ondwa setup` command.



Important: After initial setup, if you update any of the parameters in the `dwainst.conf` configuration file, you must run the following `ondwa` commands again for the updated parameters to take effect:

- `ondwa stop`
- `ondwa setup`
- `ondwa start`

The `dwainst.conf` file contains the following parameters.

Table 8. Parameters in the `dwainst.conf` file

Parameter	Description	Guidance
CLUSTER	For cluster installations:	Common examples are <code>eth0</code> , <code>eth1</code> , or <code>eth2</code> .
R_INTER_FACE	The network device name for the connection between the cluster nodes.	
COORDINATOR_SHM	The value, in megabytes, of the shared memory on the coordinator node.	The total of the shared memory on the coordinator node and worker nodes should not exceed the free memory on the computer where the accelerator server is installed.
<p> Tip: The coordinator node does not need as much shared memory as the worker nodes. A value between 5 to 10 % of the total memory set aside for the accelerator server is a good estimate for this parameter.</p>		
CORES_FOR_LOAD_THREATS_PERCENTAGE	Used in cluster installations.	If only one coordinator node or one worker node will run on each cluster node, set this value to <code>100</code> .
CORES_FOR_REPLICATION	Used in cluster installations.	If only one coordinator node or one worker node will run on each cluster node, set this value to <code>25</code> .

Table 8. Parameters in the dwainst.conf file (continued)



Parameter	Description	Guidance
ORG_TH READS_ PERCENT TAGES		
CORES_ FOR_SC AN_THR EADS_P ERCENT AGE	Used in cluster installations.	If only one coordinator node or one worker node will run on each cluster node, set this value to 100.
DRDA_IN TERFACE E	The network device name that you will use for the connection from the Informix® database server to the accelerator server. The default value is 10.	Ask your system administrator and network administrator which network interface to use for the DRDA_INTERFACE value. If the accelerator server is installed on a separate computer than the Informix® database server, you cannot use the local loopback value.
DWADIR	The name and file path for accelerator server storage directory.	Create the directory first and then specify the directory in the dwainst.conf file.  Note: Specify the directory before you run the ondwa setup command.
NUM_N ODES	The number of nodes (DWA_CM processes)	The number of accelerator server nodes should not overload the computer where the accelerator server is installed. The number of worker nodes is the value of the NUM_NODES parameter - 1.
START_ PORT	The starting port number for the coordinator node and the worker nodes.	The accelerator server assigns the port numbers that immediately follow the starting port number to the coordinator node and the worker nodes. These port numbers should not already be used by other processes. Each accelerator server node needs to be configured with four different port numbers. Beginning with the starting port number, the nodes are assigned incremental port numbers. For example, if your accelerator server has five nodes and you specify the START_PORT number as 21020, the accelerator server will use ports 21020 - 21039 because each node uses four port numbers.
WORKER _SHM	The value, in megabytes, of the total shared	The combined total shared memory on the worker nodes and the coordinator node should not exceed the free memory on the computer where the accelerator server is installed.

Table 8. Parameters in the dwainst.conf file (continued)

Parameter	Description	Guidance
	memory, combined, on all the worker nodes.	 Tip: The data marts, with all their data in the compressed format, must fit into the shared memory on the worker nodes. Plan on using approximately two-thirds of the total memory for the accelerator server as worker nodes shared memory.

The following example shows parameters and their values in a `dwainst.conf` configuration file:

```
DWADIR=$IWA_INSTALL_DIR/dwa/demo
START_PORT=21020
NUM_NODES=2
WORKER_SHM=500
COORDINATOR_SHM=250
DRDA_INTERFACE="eth0"
```

Related reference

[ondwa setup command on page 88](#)

Related information

[Informix Warehouse Accelerator directory structure on page 71](#)

[Configuring memory for Informix Warehouse Accelerator on page 83](#)

[Configuring Informix Warehouse Accelerator on page 76](#)

[Configuring Informix Warehouse Accelerator for hardware clusters on page 77](#)

Configuring memory for Informix® Warehouse Accelerator

To process queries efficiently, verify that you have the optimal configuration for the operating-system kernel parameters, shared memory, and swap space on the computer where is installed. You can also monitor the virtual memory usage to see if you need to reduce the total size of the loaded data marts or add physical memory.

[vm.overcommit_memory and the vm.overcommit_ratio kernel parameters on page 83](#)

[SHMMAX kernel parameter on page 84](#)

[Shared memory for the worker nodes and the coordinator node \(/dev/shm\) on page 84](#)

[Swap space configuration on page 85](#)

[Virtual memory usage on page 85](#)

vm.overcommit_memory and the vm.overcommit_ratio kernel parameters

To avoid issues that might arise if the Linux™ kernel runs out of memory, set the `vm.overcommit_memory` parameter and the `vm.overcommit_ratio` parameter to values that are optimal for .

1. Add the following lines to the `/etc/sysctl.conf` file:

```
vm.overcommit_memory = 2
vm.overcommit_ratio = 99
```

2. Run the `sysctl -p` command for the settings to take effect.

For more information, see your operating-system documentation.

SHMMAX kernel parameter

The SHMMAX kernel parameter defines the maximum size in bytes of a single shared memory segment. For , the optimal value of the SHMMAX kernel parameter is the size of physical memory. The minimum value is the `WORKER_SHM` parameter * 1048576 .

Use this command to check the value of the SHMMAX kernel parameter. The output value is printed in bytes.

```
sysctl kernel.shmmax
```

To change the value of the SHMMAX kernel parameter:

1. Add the following line to `/etc/sysctl.conf` file where *bytes* is the number of bytes:

```
kernel.shmmax = bytes
```

2. Run the `sysctl -p` command for the setting to take effect.

For more information, see your operating-system documentation.

Shared memory for the worker nodes and the coordinator node (/dev/shm)

The shared memory is used to hold the data mart data. The more worker nodes that you designate, the faster the data is loaded from the database server. However, the more worker nodes you designate, the more memory you need because each worker node stores a copy of the data in the dimension tables. If you do not have sufficient memory assigned to the coordinator node and to the worker nodes, you might receive errors when you load data from the database server.

uses the shared memory (`/dev/shm`) for the shared memory of the coordinator node and the worker nodes.

The default size of `/dev/shm` is half of the total memory. If you change the size of `/dev/shm`, it must be smaller than the total memory in order to leave enough memory for other tasks.

For installed on a single computer, the sum of values for the `WORKER_SHM` parameter and the `COORDINATOR_SHM` parameter must fit in available space of `/dev/shm`.

For installed on a hardware cluster, the value of the `WORKER_SHM` parameter/ (`NUM_NODES` parameter - 1) must fit in available space of `/dev/shm` on each worker node. The value of the `COORDINATOR_SHM` parameter must fit in available space of `/dev/shm` on the coordinator node.

You specify the values for the `WORKER_SHM` parameter and the `COORDINATOR_SHM` parameter in the `dwainst.conf` configuration file.

To check the size of the available space in `/dev/shm`, run this command:

```
df -h /dev/shm
```

Example output:

```
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           1.0G   0  1.0G   0% /dev/shm
```

To change the size of `/dev/shm`, use the `tmpfs` mount option `"size=nbytes"`. Use the suffix `k` for kilobytes, `m` for megabytes, or `g` for gigabytes.

Example line in the file-system table file `/etc/fstab` with the size of `/dev/shm` set to 1 gigabyte:

```
tmpfs /dev/shm tmpfs defaults,size=1g 0 0
```

The new size is effective when you remount `/dev/shm` or when you restart the computer. For example:

```
mount -o remount /dev/shm
```

Swap space configuration

Even on systems that have a large amount of total memory, configuring the swap space can be an advantage. Swap space can prevent unexpected situations where the system might need more memory than is available.

Check the total and the used swap space with this command:

```
free -m
```

Example output (in megabytes):

```
Mem:          total      used      free     shared    buffers     cached
-/+ buffers/cache:    182      1827
Swap:         3138         0       3138
```

Configure the swap space as recommended by your Linux™ vendor. For example, for Red Hat Enterprise Linux™ (RHEL) 5:

Total memory	Total swap space
4GB or less	min 2GB
4GB to 16GB	min 4GB
16GB to 64GB	min 8GB
64GB to 256GB	min 16GB
256GB to 512GB	min 32GB

For more information, see your operating-system documentation.

Virtual memory usage

Some types of SQL queries might need a lot of memory, which is allocated on an as-needed basis. This additional memory is different from the shared memory that is used for the data mart data. For optimal performance, avoid swapping out memory pages to disk. First check if memory pages are currently swapped out to disk. If memory pages are swapped out to disk, reduce the total size of the loaded data marts or add physical memory.

Check if the memory pages are swapped out to disk with this command:

```
vmstat -S m 3
```

Example output in megabytes, and 3-seconds intervals:

```
procs -----memory----- ---swap-- ----io---- --system-- -----cpu-----
r  b  swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa  st
0  0    54  1843   3   200   0   0   17  20  14   9  2  0  97  0  0
1  0    54  1673   3   201   0   0  263   0 1028 193 11  3  85  2  0
0  0    54  1803   3   201   0   0   0  36 1019 222 35  9  56  0  0
```

If values in `so` column are greater than zero, then pages are swapped out in the interval.

An alternative method for automated monitoring is this command:

```
vmstat -s -S m
```

Example output (in megabytes, swap-related lines only):

```
201 m swap cache
3291 m total swap
54 m used swap
3237 m free swap
6761440 pages swapped in
7111235 pages swapped out
```

If value of `pages swapped out` has incremented since the last time that you ran the command, then the pages have been swapped out.

For more information, see your operating-system documentation.

Related reference

[dwainst.conf configuration file on page 80](#)

Related information

[prerequisites and requirements on page 70](#)

[Informix Warehouse Accelerator internal architecture on page 58](#)

The ondwa utility

Use the `ondwa` utility to set up and work with the accelerator server.

Prerequisites

The `ondwa` utility is a Bash shell script. Because the accelerator server is supported only on Linux™ operating systems, the ability to run Bash shell scripts is built into the operating system. The following prerequisites must be installed on the machine where you installed the accelerator server:

- Telnet client program
- Expect utility
- `su` command
- User `informix` must exist

The ondwa utility directory

The `ondwa` utility is located in the `$IWA_INSTALL_DIR/bin` directory.

Running the ondwa utility in a cluster system

If you have installed on a cluster system, you can run the `ondwa` command from any cluster node. The `ondwa` command will run on all cluster nodes listed in the `$IWA_INSTALL_DIR/dwa/etc/cluster.conf` file.

Related information

[Configuring Informix Warehouse Accelerator on page 76](#)

Users who can run the ondwa commands

Either user root or user informix can run the ondwa commands. To run the ondwa commands as user informix requires setup steps. It is recommended that you determine which user will run the ondwa commands and to use the same user consistently. Whether you run the ondwa commands as user root or as user informix, user informix must be defined on the accelerator server. Otherwise, the ondwa commands will fail.

To run ondwa commands as user informix, the following restrictions apply:

- The directory specified in the DWADIR parameter in the `dwainst.conf` file must be owned by user informix.
- If you have a `DWA_watchdog.log` file, it must be writable for user informix. The `DWA_watchdog.log` file is in the directory specified by the DWADIR parameter.
- The following shell soft and hard limits must be set to unlimited prior to running ondwa commands. For example, you can use the built-in Linux™ Bash shell `ulimit` command to change the resource availability.

Table 9. Resources that must be set to unlimited for user informix to run ondwa commands.

Resource	Bash sell ulimit command
max locked memory	<code>ulimit -l</code>
max memory size	<code>ulimit -m</code>
virtual memory	<code>ulimit -v</code>

If you installed on a cluster system with user informix, you can set the following equivalent parameters to unlimited in the `/etc/security/limits.conf` file on each cluster node:

memlock

max locked-in-memory address space

rss

max resident set size

as

address space limit

For example:

```
informix    soft    memlock    unlimited
informix    hard    memlock    unlimited
informix    soft    rss        unlimited
```

informix	hard	rss	unlimited
informix	soft	as	unlimited
informix	hard	as	unlimited

ondwa setup command

The `ondwa setup` command creates the files and subdirectories that are required to run an accelerator server.

Edit the `dwainst.conf` configuration file before you run the `ondwa setup` command. The `dwainst.conf` configuration file contains information that is used to configure .

To run this command, log on to the computer where the accelerator server is installed either as user `root` or as user `informix`. For you to run the command as user `informix`, specific memory resources must be available. For more information, see [Users who can run the ondwa commands on page 87](#).

Usage:

```
$ ondwa setup
```

The `ondwa setup` command uses a file named `dwainst.conf` in the `$IWA_INSTALL_DIR/dwa/etc` directory to configure the accelerator server.

By using the `dwainst.conf` file, the `ondwa setup` command creates the following structure in the accelerator server storage directory:

- The directory shared between the accelerator server nodes (`shared`)
- The directory that contains the accelerator server node private directories (`local`)
- For each accelerator server node:
 - The accelerator server node private directory: `local/node`
 - The accelerator server node configuration file: `node.conf`
 - A link to the `DWA_CM` executable file: `DWA_CM_node`

The `ondwa` utility automatically determines the role of the coordinator node or a worker node. The first node is the coordinator node, whereas the remaining nodes are worker nodes. The number of worker nodes is determined by the following calculation: `NUM_NODES - 1`.

Each accelerator server node needs to be configured with four different port numbers. These port numbers are listed in the configuration file for the accelerator server node. The starting port number is taken from the `dwainst.conf` file, and then increased by increments. For example, if your accelerator server has five accelerator server nodes and you specify the `START_PORT` number as 21020, the accelerator server will use ports 21020 - 21039 because each accelerator server node uses four port numbers.

When an accelerator server node is started, the corresponding link to the accelerator server binary `DWA_CM_node` is used. The `ondwa setup` command creates a symbolic link for each accelerator server node to the `DWA_CM` binary. The link makes it easier to find the processes of your accelerator server in a `ps` output because the `ps` command shows the symbolic links and not the `DWA_CM` binary itself.

Related reference[ondwa start command on page 89](#)[dwainst.conf configuration file on page 80](#)**Related information**[Configuring Informix Warehouse Accelerator on page 76](#)

ondwa start command

The `ondwa start` command starts all of the accelerator server nodes. If the accelerator server is installed on a cluster hardware system, and if some of the cluster nodes are stopped, the `ondwa start` command starts the offline cluster nodes.

To run this command, log on to the computer where the accelerator server is installed either as user `root` or as user `informix`. For you to run the command as user `informix`, specific memory resources must be available. For more information, see [Users who can run the ondwa commands on page 87](#).

Usage:

```
$ ondwa start
```

The output of a accelerator server node is recorded in the log file for the node, for example: `node0.log`, `node1.log`, and so forth. These files are located in the accelerator server storage directory. After the `ondwa start` command has finished, your accelerator server is ready to use.

After you run `ondwa start`, you should run the `ondwa status` command to check that status of the accelerator server.

Related reference[ondwa setup command on page 88](#)[ondwa status command on page 89](#)**Related information**[Configuring Informix Warehouse Accelerator on page 76](#)

ondwa status command

The `ondwa status` command shows information about the status of the accelerator server nodes, the cluster status, and the expected node count.

To run this command, log on to the computer where the accelerator server is installed either as user `root` or as user `informix`. For you to run the command as user `informix`, specific memory resources must be available. For more information, see [Users who can run the ondwa commands on page 87](#).

Usage:

```
$ ondwa status
ID | Role          | Cat-Status | HB-Status | Hostname | System ID
-----+-----+-----+-----+-----+-----
0  | COORDINATOR | ACTIVE     | Healthy   | leo     | 1
1  | WORKER      | ACTIVE     | Healthy   | leo     | 2
Cluster is in state : Fully Operational
Expected node count : 1 coordinator and 1 worker nodes
```

The following example output shows the status shows one accelerator server node is shut down:

```
$ ondwa status
ID | Role          | Cat-Status | HB-Status | Hostname | System ID
-----+-----+-----+-----+-----+-----
0  | COORDINATOR | ACTIVE     | Healthy   | leo     | 1
1  | WORKER      | DEACTIVATED | Shutdown  | leo     | 2
Cluster is in state : Recovering
Expected node count : 1 coordinator and 1 worker nodes
```

The following example output shows the status of a cluster system:

```
ID | Role          | Cat-Status | HB-Status | Hostname | System ID
-----+-----+-----+-----+-----+-----
0  | COORDINATOR | ACTIVE     | Healthy   | node0001 | 1
1  | WORKER      | ACTIVE     | Healthy   | node0002 | 2
2  | WORKER      | ACTIVE     | Healthy   | node0003 | 3
3  | WORKER      | ACTIVE     | Healthy   | node0004 | 4
Cluster is in state      : Fully Operational
Expected node count     : 1 coordinator and 3 worker nodes
```

Columns in ondwa status command output

The `Cat-Status` column shows the status of all nodes in the accelerator server catalog. The `Cat-Status` column can have one of the following values:

ACTIVE

The accelerator server node is active. For a cluster system, `ACTIVE` indicates that the cluster node participates in the cluster.

DEACTIVATED

The accelerator server node is shut down or the accelerator server is not running.

FAILOVER

A failed accelerator server node that was set in ERROR state.

The `HB-Status` (*heartbeat* status) column can have one of the following values:

Initializing

The accelerator server node is initializing and loading data into memory.

Healthy

The initialization is complete and the accelerator server node is ready for queries.

QuiescePend

The accelerator server node is waiting to go into a quiesced state.

Quiesced

The accelerator server node is in a quiesced state.

Resuming

The accelerator server node is resuming after a quiesced or a maintenance state.

Maintenance

The accelerator server node is in maintenance state.

MaintPend

The accelerator server node is waiting to go into a maintenance state.

Missing

The accelerator server node has a missing heartbeat.

Shutdown

The accelerator server node is shutting down.

Related reference

[ondwa start command on page 89](#)

ondwa listmarts command

Use the `ondwa listmarts` command to show information about existing data marts in the accelerators.

To run this command, log on to the computer where the accelerator server is installed either as user `root` or as user `informix`. For you to run the command as user `informix`, specific memory resources must be available. For more information, see [Users who can run the ondwa commands on page 87](#).

Usage:

```
$ ondwa listmarts
```

The following example lists the existing data marts that are found in the accelerators:

```
Found 4 marts:
mart="cq_sqrt", martid="151", accelerator="IDS.dwa1", state="ACTIVE",
  epoch="1", lastload="2014-12-08 11:23:16.788831"
mart="iwadb_mart1", martid="152", accelerator="IDS.dwa1", state="ACTIVE",
  epoch="3", lastload="2015-01-09 17:44:40.822332",
  lastupdate="2015-01-13 16:43:49.048554"
mart="iwadb_mart2", martid="153", accelerator="IDS.dwa1", state="LOAD PENDING",
  epoch="0"
mart="iwadb_mart2", martid="154", accelerator="IDS.dwa2", state="LOAD PENDING",
  epoch="0"
```

ondwa getpin command

Use the `ondwa getpin` command to retrieve the IP address, port number, and pairing code that connect a database server to an accelerator server.

To run this command, log on to the computer where the accelerator server is installed either as user `root` or as user `informix`. For you to run the command as user `informix`, specific memory resources must be available. For more information, see [Users who can run the ondwa commands on page 87](#).

Usage:

```
$ ondwa getpin
```

The following example shows the values that are returned in this order: IP address, port number, pairing code.

```
127.0.0.1 21022 1234
```

ondwa tasks command

The `ondwa tasks` command shows information about the tasks that are currently running, memory that is used, and resources.

To run this command, log on to the computer where the accelerator server is installed either as user `root` or as user `informix`. For you to run the command as user `informix`, specific memory resources must be available. For more information, see [Users who can run the ondwa commands on page 87](#).

The `ondwa tasks` command shows information about the following types of tasks:

- **GENERIC:** An unknown or generic task
- **QUERY:** A running or queued query
- **LOAD:** LOAD data processing
- **UPDATE:** Incremental update processing, such as data mart refreshing and trickle feed
- **DAEMON:** A running daemon
- **CLIENT CONNECTION:** The AccessController establishing a connection and authenticating a user
- **TRACE COLLECTION:** Trace collection

You can identify the client session in the database server that issued a query that is being run by . The `ondwa tasks` command shows the client session number, the `@` symbol, and the value of the `SERVERNUM` configuration parameter of the database server: for example, `Session 4 @ Servernum 74`. Queries that are queued have the task name `Query execution @ coordinator`.

```

$ ondwa tasks

TaskManager tracking 3 task(s):
-----+-----+-----+-----+-----+-----+
Task 410390516044136472 (of type 'QUERY' with name Session 4 @ Servernum 74 - 0:00)
-----+-----+-----+-----+-----+-----+
Location          | Status                | Progr. | Upd. ms | Memory | Monitor
-----+-----+-----+-----+-----+-----+
Primary Node 0   | OPNQRY                | 0      | 59      | 163K   | Fine
-> Node 1        | Dim 110               | 0      | 3       | 18M    | Fine
(Total Memory)  |                       |        |         | 18M    |
-----+-----+-----+-----+-----+-----+
Used Resources   | Mart ID: 0 1 on node 0
-----+-----+-----+-----+-----+-----+
Task 227150306205499494 (of type 'QUERY' with name Query execution @ coordinator - 0:00)
-----+-----+-----+-----+-----+-----+
Location          | Status                | Progr. | Upd. ms | Memory | Monitor
-----+-----+-----+-----+-----+-----+
Primary Node 0   | Queued                | 0      | 160     | 0      | Fine
(Total Memory)  |                       |        |         | 0      |
-----+-----+-----+-----+-----+-----+
Task 437785070080 (of type 'DAEMON' with name DRDADAemon - 4:48)
-----+-----+-----+-----+-----+-----+
Location          | Status                | Progr. | Upd. ms | Memory | Monitor
-----+-----+-----+-----+-----+-----+
Primary Node 0   | Running               | 2      | 30      | 0      | Fine
(Total Memory)  |                       |        |         | 0      |
-----+-----+-----+-----+-----+-----+
Used Resources   | DRDA device: 'lo' address: '127.0.0.1:21022' on node 0
                  | Unbound on node 0
-----+-----+-----+-----+-----+-----+
- End of Tasklist -

```

ondwa stop command

The `ondwa stop` command stops the accelerator server.

To run this command, log on to the computer where the accelerator server is installed either as user `root` or as user `informix`. For you to run the command as user `informix`, specific memory resources must be available. For more information, see [Users who can run the ondwa commands on page 87](#).

Usage:

```

$ ondwa stop

$ ondwa stop -f

```

The `ondwa stop` action ends when all of the `DWA_CM_*` processes on the accelerator server are completed.

Use the `-f` option to stop the full set of `DWA_CM` processes. Use this option if the `ondwa stop` action is not able to shut down one or more of the processes.

ondwa reset command

The `ondwa reset` command removes all of the files from the accelerator server storage directory that were created by the accelerator server under the `shared` and `local/*` subdirectories.

To run this command, log on to the computer where the accelerator server is installed either as user `root` or as user `informix`. For you to run the command as user `informix`, specific memory resources must be available. For more information, see [Users who can run the ondwa commands on page 87](#).



Important: Before using this command, you must drop all data marts on the accelerator server.

Usage:

```
$ ondwa reset
```

The `ondwa reset` command also removes all of the entries of your accelerator server under the `/dev/shm` directory.

This command does not remove the log files for the accelerator server node and the files created by the `ondwa setup` command. After you run the `ondwa reset` command, you can initialize the accelerator server by using the `ondwa start` command.

ondwa clean command

The `ondwa clean` command cleans the accelerator server directory.

To run this command, log on to the computer where the accelerator server is installed either as user `root` or as user `informix`. For you to run the command as user `informix`, specific memory resources must be available. For more information, see [Users who can run the ondwa commands on page 87](#).



Important: Before using this command, all of the data marts must be dropped in the accelerator and the accelerator server must be removed from the computer.

Usage:

```
$ ondwa clean
```

The `ondwa clean` command removes the following files and directories:

- All of the files created by the `ondwa setup` command
- The complete `shared` and `local` directory trees in the accelerator server storage directory
- The log files for the accelerator server node

After you run the `ondwa clean` command, you can run the `ondwa setup` command to set up the accelerator server again.

Create an accelerator

You create an accelerator by connecting the Informix® database server to the accelerator server.

An accelerator is a logical entity that contains information for a connection from the database server to the accelerator server and for the data marts that are associated with that connection. Set up the connection by using the `ondwa getpin` command to retrieve the IP address, port number, and pairing code from the accelerator server. You can then use one of the following tools to create the accelerator:

- The SQL administration [ifx_setupDWA\(\) function on page 158](#).
- Java™ classes.

Related information

[Java classes for Informix Warehouse Accelerator on page 60](#)

Data marts and AQTs

For efficient query processing, the accelerator server must have its own copy of the data. The data is stored in logical collections of related data, or data marts. A *data mart* specifies the collection of tables that are loaded into an accelerator and the relationships, or references, between these tables.

A data mart definition is an XML file that contains information about the tables used by your warehouse queries. The information includes the tables and columns within the table that are included in the data mart. The information also specifies how the tables and columns are related to each other. The XML file that contains the data mart definition does not contain any actual user data. You can create a data mart definition by analyzing the workload for your existing warehouse queries against your existing warehouse schema.

When a data mart is created, information about the data mart is sent to the Informix® database server in the form of a special view referred to as an *accelerated query table* or AQT. The information in the AQTs is used by the database server to determine which queries or query blocks can be processed by .

Accelerated query tables

Accelerated query tables, or AQTs, store information about data marts that the database server uses to determine which queries can be processed by .

The information about AQTs that is stored in the catalog tables of the database server is the same information that is stored in the catalog tables for other types of views.

There can be many AQTs. If the first AQT is not able to satisfy a query, the database server continues to search for a match until the query is checked against all of the AQTs. If a match is found, the query is sent to for processing. If no match is found, the query is processed by the database server.

A single AQT can have a maximum of 255 tables and 750 columns.

To be sent to , the query must meet the following criteria:

- The query must refer to a subset of the tables in the AQT.
- The table references, or joins, specified in the query must be the same as the references in the data mart definition.
- The query must include only one fact table.
- The query must have an INNER JOIN or LEFT JOIN with the fact table on the left dominant side.
- The scalar and aggregate functions in the query must be supported by .

A data mart can be in one of several different states, but the associated AQTs are either active or inactive.

When you drop a data mart from an accelerator, the associated AQTs are removed automatically from the database server.

You can use the `onstat -g aqt` command to view information about the data marts and the associated AQTs.

Related information

[Queries that benefit from acceleration on page 61](#)

Data marts

Typically, data marts contain a subset of the tables in your database. The data marts can also contain a subset of the columns within a table. This configuration is advantageous when you are using the TCP/IP loopback optimization between Informix® and IWA, because it provides a seamless experience for the customer.

When you create a data mart, you specify the fact table, the dimension tables, and the references between the tables. Data marts do not need to be a duplication of the design of your warehouse fact and dimension tables. For example, you can designate a dimension table in your warehouse schema as a fact table in a data mart. To improve query processing, limit the number of dimension tables, and columns within the dimension tables, in the data mart. Identify only those columns that are necessary to respond to your queries.

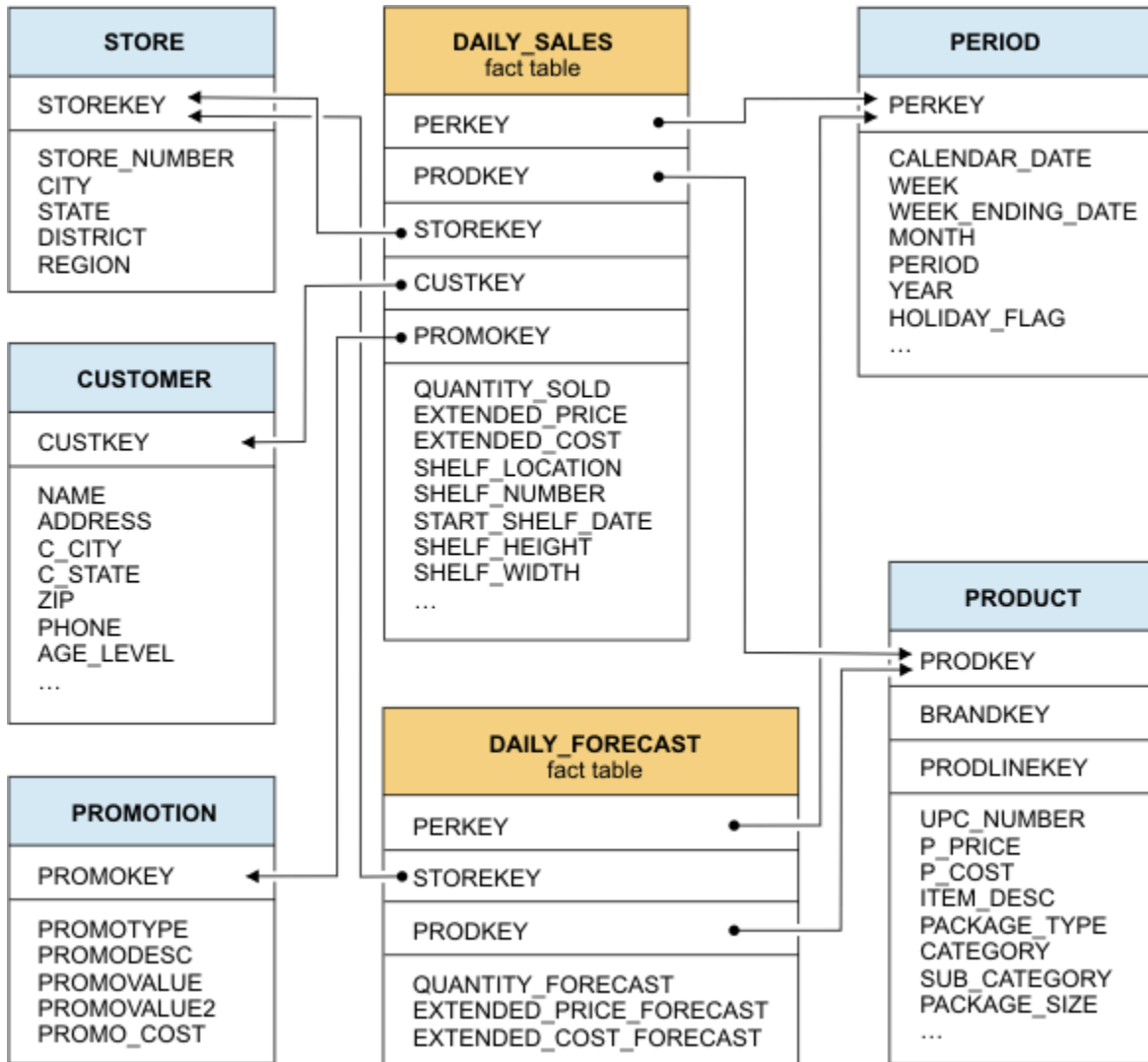
If a fact or dimension table is fragmented, each fragment corresponds to a partition in the data mart. If a fact or dimension table is not fragmented, the data mart contains a single partition that contains the data from the table.

A newly created data mart has all of the necessary structures that are defined but is empty and must be filled with a snapshot of the data from the Informix® database server. When the data from the database server is loaded in the data mart in the accelerator server, the data is compressed. After the data is loaded in the data mart, the data mart becomes operational.

Data marts must be based on a snowflake or star schema

The following figure shows a sample schema with two fact tables, DAILY_SALES and DAILY_FORECAST. These fact tables are linked to several dimension tables: STORE, CUSTOMER, PROMOTION, PERIOD, and PRODUCT. There are several key references in the fact tables that are used to link to the dimension tables. For example, in the DAILY_SALES fact table, the PRODKEY column is linked to the PRODKEY column in the PRODUCT dimension table.

Figure 19. A sample star schema with two fact tables



Using the schema in [Figure 19: A sample star schema with two fact tables on page 97](#), you can create two data marts. The first data mart is based on the DAILY_SALES fact table and the dimension tables that it links to, as shown in [Figure 20: A data mart with the DAILY_SALES fact table on page 98](#). A second data mart is based on the DAILY_FORECAST fact table and the dimension tables that it links to, as shown in [Figure 21: A data mart with the DAILY_FORECAST fact table on page 99](#).

Figure 20. A data mart with the DAILY_SALES fact table

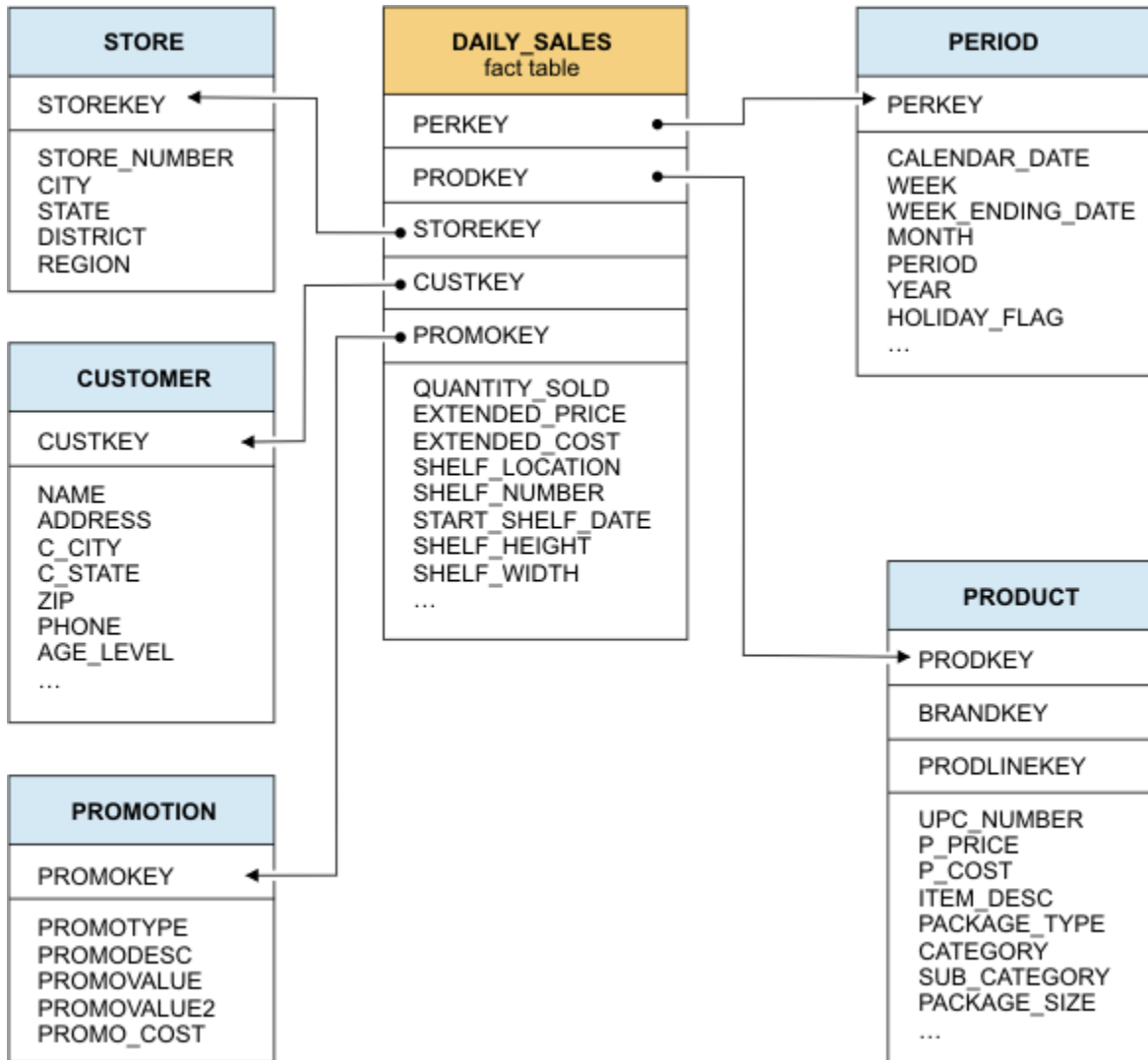
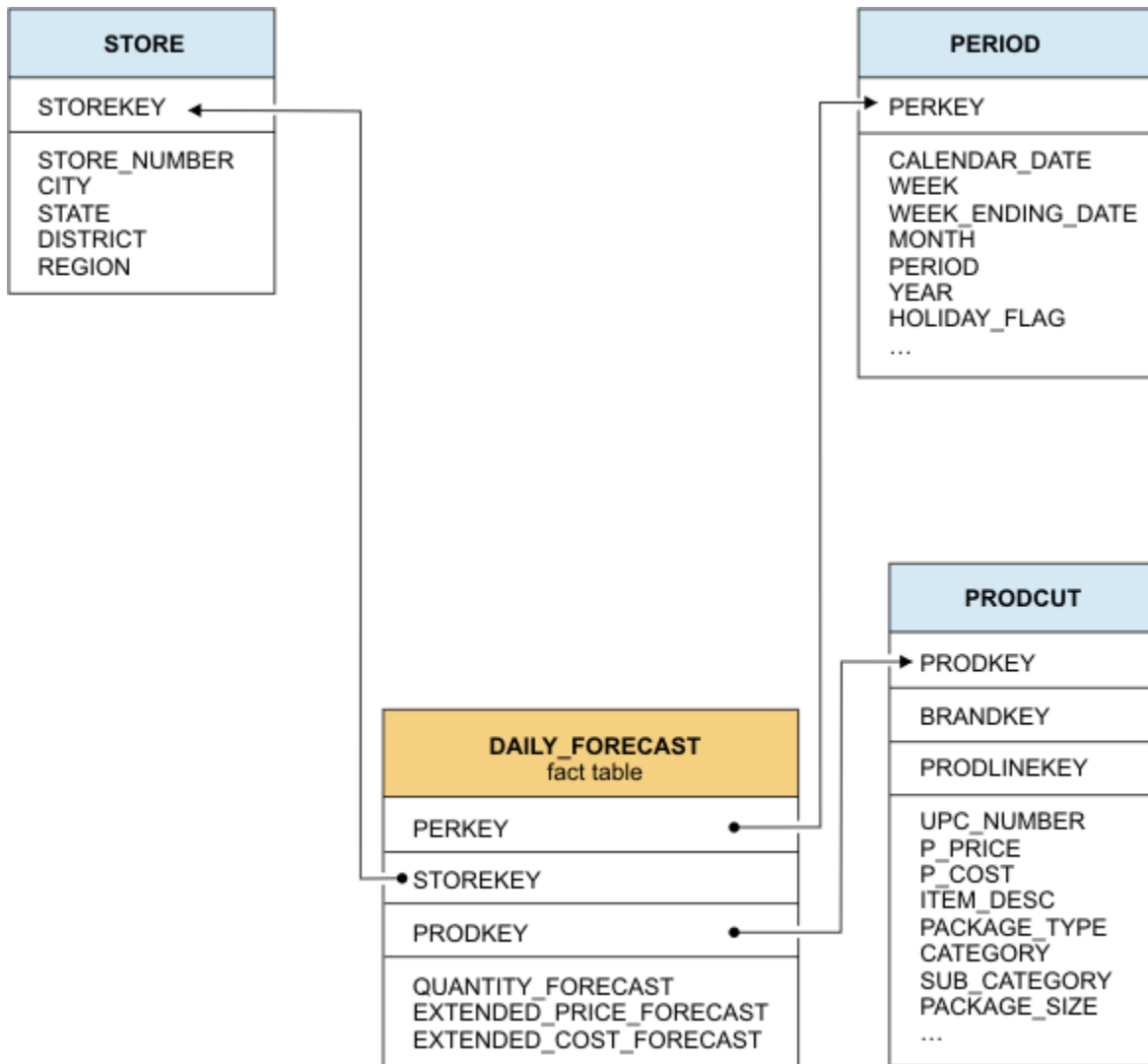


Figure 21. A data mart with the DAILY_FORECAST fact table



Summary or aggregate tables in data marts

To summarize the granular data in the fact tables and dimension tables, some warehouse databases use other tables that are known as *summary tables* or *aggregate tables*. For example, a summary table might contain sales information for an entire month or quarter that is consolidated from fact and dimension tables.

Because speeds up query processing, it is not necessary to use summary tables to improve query performance. queries the fact table and dimension tables directly.

Create a data mart

After you configure the accelerator server and set up the connection between an accelerator and the database server, you need to create the data marts.

A data mart specifies the collection of tables from a database that is loaded into an accelerator and the relationships, or references, between these tables. To create the data mart, you must determine which warehouse queries are good candidates for acceleration.

Create a data mart by using one of the following tools:

- The SQL administration [ifx_createMart\(\) function on page 136](#).
- Manual workload analysis requires that you run a series of statements, stored procedures, and functions that analyze the schema and the queries to create a data mart definition. After you create the data mart definition, you must deploy it to create the data mart.
- Java™ classes.

Related information

[Creating data mart definitions manually by using workload analysis on page 101](#)

[Java classes for Informix Warehouse Accelerator on page 60](#)

Workload analysis

Workload analysis is a process of designing an effective data mart.

You can do the workload analysis manually.

To get the best performance from accelerated queries, you need to have an optimal data mart that includes all the necessary columns and tables, but excludes any columns and tables that are not used in your warehousing queries. The data mart must also specify the necessary joins between the tables as they are used in the queries. If your data warehouse has a complex database schema and a workload that includes reports that might run hundreds of different queries, it is challenging to determine the optimal data mart definition. Workload analysis gathers information about your warehousing queries as you run them, known as query probing, and analyzes the resulting data to determine which queries are good candidates for acceleration. The columns, tables, and joins used by the queries that can be accelerated are included in the data mart definition.

Workload analysis needs to determine the fact table of a query. If parallel database query (PDQ) is active, you can specify the fact table with the FACT optimizer directive. If the FACT optimizer directive is not set, and for inner join queries, the fact table is identified as the table with the most number of rows.

By default, information from all queries that you run during query probing is included in the data mart definition. You can customize the data mart definition by including specific queries in the definition. If you turn SQL tracing on, you can identify the probing data that resulted from a specific SQL statement by its statement ID.

If you do workload analysis manually, you can identify specific statements, for example statements that took a certain length of time to process, or statements that accessed specific tables. With that information, you can include the probing data that resulted from only these statements in the mart definition. To view SQL tracing information, run the `onstat -g his` command or query the `syssqltrace` table in the `sysmaster` database. For example, in DB-Acess, run this SQL statement:

```
SELECT sql_runtime, sql_statement FROM syssqltrace
WHERE sql_stmntname matches "SELECT" ORDER BY sql_runtime DESC
```

By default, query probing is based on the running time of each query. To run the query probing more quickly, issue the `SET EXPLAIN ON AVOID_EXECUTE` statement before you run your query workload. When you issue this statement, the queries are optimized and the probing data is collected, but a result set is not determined or returned. However, if you avoid running the queries, you do not know exactly how long it takes to run the queries.

The query probing data is stored in memory. You can view the query probing data by running the `onstat -g probe` command or by querying the system monitoring interface.

Creating data mart definitions manually by using workload analysis

Workload analysis involves two main steps, gathering information about your query workload, which is known as query probing, and analysis of the query probing data.

Before you begin

Prerequisites:

- You must have an existing database with a star or snowflake schema on a local database of the server that you are connected to.
- You must have warehousing queries that match the acceleration criteria.
- A default sbspace must exist.

To create a data mart definition by using workload analysis:

1. Connect to the database that contains the data warehouse tables.

In these instructions, this database is referred to as the warehouse database.

2. Update database statistics with the `LOW` option to generate a minimum amount of statistics for the database by running the `UPDATE STATISTICS LOW` statement.

Query probing is required to determine the fact table of the queries being analyzed. In many cases, the fact table is defined by the query (ANSI JOIN syntax), by star join optimization, or by optimizer directives that are provided by the user. In other cases, the table that contains the most rows is defined as the fact table. Running the `UPDATE STATISTICS LOW` statement ensures that database statistics contain the correct number of rows per table.

3. Remove existing probing results for the current database by running the following statement:

```
SET ENVIRONMENT use_dwa 'probe cleanup';
```

4. Enable query probing for the current user session by running the following statement:

```
SET ENVIRONMENT use_dwa 'probe start';
```

5. **Optional:** If you want to customize your data mart definition to include only the probing results of the queries run in step 4 on page 101, enable SQL tracing by running the following statement as user **informix** connected to the **sysadmin** database:

Example

```
EXECUTE FUNCTION task("set sql tracing on","1000","4","low","global");
```

6. **Optional:** To run the query probing more quickly by simulating running the query workload, run the following statement:

```
SET EXPLAIN ON AVOID_EXECUTE;
```

Simulating the query workload does not provide results.

7. Run your warehousing queries in the current user session.
8. Disable query probing for the current user session by running the following statement:

```
SET ENVIRONMENT use_dwa 'probe stop';
```

9. **Optional:** If you enabled SQL tracing, view the SQL trace information about the workload by either running the `onstat -g his` command or by querying table **syssqltrace** in the **sysmaster** database.
10. If required, create a new logged database to store the data mart definition.

The data mart definition is stored in a set of tables called the data mart schema tables. These tables must reside in a logged database. The data mart schema tables are created automatically if they do not exist. Any logged database can contain them, including the warehouse database assuming it is a logged database. To separate the data mart schema tables from other user tables, create a separate database for them. This database is referred to as the schema database in the steps that follow.

11. Convert the probing data into a data mart definition by running the `ifx_probe2Mart()` procedure in the schema database.

Choose from:

- To create a data mart definition from all of the probing data, run the following statement:

```
EXECUTE PROCEDURE ifx_probe2mart('warehouse database', 'data mart name');
```

- To create a data mart definition or extend an existing data mart definition by using the probing data of a single query, run the following statement:

```
EXECUTE PROCEDURE ifx_probe2mart('warehouse database', 'data mart name', sql id);
```

To refer to a specific query by its *sql id*, SQL tracing must be enabled, as described in 5 on page 102.

12. **Optional:** Remove the probing data from memory before you start any new query probing sessions by running the following command in your warehouse database:

```
SET ENVIRONMENT use_dwa 'probe cleanup';
```

13. **Optional:** You can view the data mart definition by running the `ifx_genMartDef()` function in the schema database. This function returns a CLOB that contains the data mart definition in XML format. You can store the CLOB in an operating system file on the client computer by using the `lotofile()` function:

```
EXECUTE FUNCTION lotofile(ifx_genmartdef('data mart name', 'file name', 'client'));
```

14. Deploy the data mart to your accelerator by using either:

Choose from:

- The `ifx_createMart()` function in your warehouse database by running the following statement:

```
EXECUTE FUNCTION ifx_createMart('accelerator name', 'data mart name', 'schema database name');
```

- The file created in [10 on page 102](#) to deploy the data mart to your accelerator by running the following statement:

```
EXECUTE FUNCTION ifx_createMart('accelerator name', filetoclob('file name', 'client'));
```

15. Load the data mart on your accelerator by using the `ifx_loadMart()` function in your warehouse database by running the following statement:

```
EXECUTE FUNCTION ifx_loadMart('accelerator name', 'data mart name', 'locking mode');
```

Related reference

[Sysmaster interface \(SMI\) pseudo tables for query probing data on page 174](#)

[use_dwa session environment variable on page 130](#)

Related information

[Deploy and load the data mart on page 109](#)

[Java classes for Informix Warehouse Accelerator on page 60](#)

Example: Create a data mart from the command line

Use this step-by-step example as a guide to create, deploy, and load a data mart from the command line using workload analysis.

This example uses the **stores_demo** database that is created by the command `dbaccessdemo`.

The workload is from the following query:

```
SELECT {+ FACT(orders)} first 5 fname,lname,sum(ship_weight)
FROM customer c,orders o
WHERE c.customer_num=o.customer_num
and state='CA' and ship_date is not null
GROUP BY 1,2
ORDER BY 3 desc;
```

The query selects the names of the top five customers from the state CA and the total ship weight of their already shipped orders. The query is an inner join. The **orders** table is the fact table. The **customer** table is the dimension table. Since the **orders** table has fewer rows than the **customer** table, the `{+ FACT(orders)}` optimizer hint is required. Otherwise, the **customer** table would be considered as the fact table.

The following commands correspond to steps in the task [Creating data mart definitions manually by using workload analysis on page 101](#).

The SQL statements used in this example are run in the DB-Access utility, and are prompted by `>`. Commands that are run from the shell are prompted by `$`.

Step 1: Connect to the database

Connect to the database. This example uses the **stores_demo** database:

```
$ dbaccess stores_demo -
Database selected.
```

Step 2: Update statistics

Update the statistics on the database:

```
> update statistics low;
Statistics updated.
```

Step 3: Remove probing results:

Remove previously existing probing results:

```
> set environment use_dwa 'probe cleanup';
Environment set.
```

Step 4: Start probing

Enable query probing:

```
> set environment use_dwa 'probe start';
Environment set.
```

Step 5: Optional - Enable SQL tracing

In a separate session, connect as user `informix` to the **sysadmin** database and activate SQL tracing:

```
$ dbaccess sysadmin -
> execute function task("set sql tracing on","1500","4","low", "global");
(expression) SQL Tracing ON: ntraces=1500, size=4056, level=Low, mode=Global.
1 row(s) retrieved.
```

Step 6: Skip running the query workload

When you issue this statement, the queries are optimized and probed but a result set is not determined or returned.



Important: If you want to process the probing data based on the run time of the queries, then turn on SQL tracing and do not use the `AVOID_EXECUTE` option of the `SET EXPLAIN` statement. If you avoid running the queries, you do not know the actual time the queries run.

```
> set explain on avoid_execute;
Explain set.
```

Step 7: Run the query workload

Using this example, the SQL statements are:

```
> SELECT {+ FACT(orders)} first 5 fname,lname,sum(ship_weight)
FROM customer c,orders o
WHERE c.customer_num=o.customer_num and state='CA'
and ship_date is notnull
GROUP BY by 1,2
ORDER BY by 3 desc;
fname          lname          (sum)
No rows found.
```

The reason no rows are returned in this example is that the SET EXPLAIN ON AVOID_EXECUTE statement has been used.

Step 8: Stop probing

Disable query probing:

```
> set environment use_dwa 'probe stop';
Environment set.
```

Step 9: Optional - View the SQL trace information

You can use the onstat command or query the SMI tables to view the SQL trace information.

To use the onstat command:

```
$ onstat -g his

HCL Informix Dynamic Server Version 11.70.FC3 -- On-Line -- Up 00:53:06 --
182532 Kbytes

Statement history:

Trace Level          Low
Trace Mode           Global
Number of traces     1500
Current Stmt ID      2
Trace Buffer size     4056
Duration of buffer   42 Seconds
Trace Flags          0x00001611
Control Block        0x4dd51028

Statement # 2:      @ 0x4dd51058

Database:           0x100153
Statement text:
select {+ FACT(orders)} first 5 fname,lname,sum(ship_weight)
from customer c,orders o
where c.customer_num=o.customer_num and state='CA' and
ship_date is not null group by 1,2 order by 3 desc

Statement information:
Sess_id  User_id  Stmt Type      Finish Time    Run Time    TX Stamp  PDQ
-----  -
51       29574   SELECT         10:39:09      0.0000     33f4e    0

Statement Statistics:
Page      Buffer    Read      Buffer    Page      Buffer    Write
Read     Read     % Cache  IDX Read  Write     Write     % Cache
-----  -
0         0        0.00     0         0         0         0.00

Lock      Lock     LK Wait   Log       Num       Disk     Memory
Requests Waits    Time (S) Space    Sorts    Sorts    Sorts
-----  -
0         0        0.0000   0.000 B  0         0         0

Total    Total    Avg       Max       Avg       I/O Wait  Avg Rows
Executions Time (S) Time (S)  Time (S) IO Wait   Time (S)  Per Sec
-----  -
1         0.0000   0.0000   0.0000   0.000000 0.000000  678122.8357
```

Estimated Cost	Estimated Rows	Actual Rows	SQL Error	ISAM Error	Isolation Level	SQL Memory
10	2	0	0	0	NL	25304

To query the SMI tables to view the SQL trace information:

```
> SELECT sql_id,sql_runtime, sql_statement
FROM sysmaster:sysssqltrace
WHERE ql_stmtname='SELECT'
ORDER BY sql_runtime desc;

sql_id          2
sql_runtime     1.47450285e-06
sql_statement   select {+ FACT(orders)} first 5 fname,lname,sum(ship_weight)
               from customer c,orders o where c.customer_num=o.customer_num
               and state='CA' and ship_date is not null group by 1,2 order by 3 desc
```

Step 10: Create the Schema database

The **stores_demo** database is not a logging database. A different database is required to store the data mart schema tables generated by running the `ifx_probe2mart()` stored procedure:

```
$ dbaccess - -
> create database stores_marts with log;
Database created.
```

Step 11: Run the `ifx_probe2mart()` stored procedure

Run the `ifx_probe2mart()` stored procedure to create the definition for data mart 'orders_customer_mart':

```
> execute procedure ifx_probe2mart('stores_demo','orders_customer_mart');
Routine executed.
```

Step 12: Optional - Remove probing data

Remove probing data gathered in the warehouse database:

```
$ dbaccess stores_demo -
> set environment use_dwa 'probe stop';
Environment set.
```

Step 13: Optional - store the data mart definition in a file

Connect to the schema database **stores_marts**, where the data mart definition that was generated by the `ifx_probe2mart()` stored procedure is stored:

```
$ dbaccess stores_marts -
> execute function lotofile(ifx_genmartdef('orders_customer_mart'),
  'orders_customer_mart.xml!','client');
(expression) orders_customer_mart.xml
1 row(s) retrieved.
```

To view the contents of the data mart definition file:

```
$ cat orders_customer_mart.xml
<?xml version="1.0" encoding="UTF-8" ?>
<dwa:martModel xmlns:dwa="http://www.ibm.com/xmlns/prod/dwa" version="1.0">
  <mart name="orders_customer_mart">
```

```

<table name="customer" schema="informix" isFactTable="false" >
  <column name="customer_num"/>
  <column name="fname"/>
  <column name="lname"/>
  <column name="state"/>
</table>
<table name="orders" schema="informix" isFactTable="true" >
  <column name="customer_num"/>
  <column name="ship_date"/>
  <column name="ship_weight"/>
</table>
<reference
  referenceType="LEFTOUTER"
  isRuntimeJoin="true"
  parentCardinality="1"
  dependentCardinality="n"
  dependentTableSchema="informix"
  dependentTableName="orders"
  parentTableSchema="informix"
  parentTableName="customer">
  <parentColumn name="customer_num"/>
  <dependentColumn name="customer_num"/>
</reference>
</mart>
</dwa:martModel>

```

Step 14: Deploy the data mart

Create the data mart **orders_customer_mart** on your accelerator **stores_accelerator**, by using schema database

stores_marts:

```

$ dbaccess stores_demo -
> EXECUTE FUNCTION ifx_createMart('stores_accelerator', 'orders_customer_mart',
  'stores_marts');
(expression) The operation was completed successfully.
1 row(s) retrieved.

```

Step 15: Load the data mart

Load the data mart **orders_customer_mart** on your accelerator **stores_accelerator**. Tables in database **stores_demo** do not become locked:

```

> EXECUTE FUNCTION ifx_loadMart('stores_accelerator', 'orders_customer_mart',
  'NONE');
(expression) The operation was completed successfully.
1 row(s) retrieved.

```

Related information

[Java classes for Informix Warehouse Accelerator on page 60](#)

Contents of query probing data

The query probing data contains a set of records when SQL tracing is used, or a single record when SQL tracing is not used.

Each record is identified by a statement ID. A record contains:

- The database name the user was connected to
- A list of tables, identified by their table id (tabid)
- A list of columns for each table, identified by their column number (colno)
- A list of references between each pair of tables, also known as the join descriptors
- A list of column pairs for each join descriptor, also known as the join predicates

A *join predicate* in the probing data corresponds to an equality join predicate between columns of different tables in the query. For example:

```
table_1.col_a = table_2.col_x
```

A *join descriptor* is comprised of:

- All of the join predicates of the same pair of tables. For example:

```
table_1.col_a = table_2.col_x and  
table_1.col_b = table_2.col_y and  
table_1.col_c = table_2.col_z
```

- The type of the join - an inner join or left outer join.
- Information about any unique indexes on the dimension table of the join. The dimension table is the right table in case of a left outer join. The unique index must be on the complete set of columns contained in this join descriptor.

Removing probing data from the database

After you no longer need the probing data, you can use a `SET ENVIRONMENT` statement to remove it from the database that contains the warehousing data. Remove probing data when you want to start a new probing session and you want to ensure that none of the previous probing data that is stored in memory is included in the new data mart definition.

About this task

To remove all of the probing data from the database:

1. Connect to the database where the warehousing data resides.
2. Run the `SET ENVIRONMENT use_dwa 'probe cleanup'` statement.

For example:

```
$ dbaccess stores_demo -  
> set environment use_dwa 'probe cleanup';  
Environment set.
```

To check if the probing data is removed, run the `onstat -g probe` command.



Important: The probing data is stored in memory. The data is automatically removed when the database server is shut down.

Related reference

[use_dwa session environment variable on page 130](#)

Deploy and load the data mart

To deploy a data mart, you specify the accelerator that will contain the data mart definition file. When you load a data mart, the related data is unloaded from the Informix® tables and transferred to the accelerator server.

When you deploy a data mart, the data mart is in the LOAD PENDING state and is disabled until you load data into the data mart.

During the loading phase, for most data marts, the data is gathered from several different tables. After the data is loaded into the data mart, it is enabled automatically and is in the ACTIVE state. Loading data into a data mart can take several hours, depending on the size and the amount of data that is contained in the tables.

Types of tables for acceleration

You can load and accelerate data from tables, external tables, synonyms, views, and time series virtual tables.

Load data from external tables

You can use external tables to load data directly from flat files or pipes to .

When you load data directly from external tables to , you are spared the interim step of loading data on to Informix® and then transferring to . When you have verified that your workload can run entirely on , you can create external tables that point to the actual data and load the data directly to . After you have created the data mart by using external tables, the queries on this data mart can run only on , not Informix®.

Loading data from external tables provides the following benefits:

- Transfer data from any source across platforms in an ASCII-delimited file to .
- Perform parallel standard INSERT operations.
- Use named pipes to support loading data from storage devices and direct network connections.
- Maintain a record of load statistics during the run.
- Perform high-speed and data-checking data load.
- Mix external and permanent tables in the data mart.

Query probing with automatic data mart generation is supported. There is minimal Informix® setup and no persistence of data in Informix®.

When loading data from external tables, the following restrictions apply:

- Locking is not supported and you must specify *locking_mode* NONE for all load operations, including `ifx_loadMart()` and `ifx_loadPartMart()`.
- If the data contains DATE or MONEY data types, the format of the data values of the fields with the DATE type must be specified during external table creation. For example, `... DBDATE 'Y4MD-' ...` must be used as table option when creating the external table if the external data (flat file or pipe) contains fields of type DATE in the format 2013-06-30.

Joins between tables can be set up among flat files or pipes. For best performance, specify a one-to-many join, which has better performance than any many-to-many join in . However, one-to-many joins require that the parent table has a unique index in place on the respective column or columns referred to in the reference definition. The existence of a unique index is verified by the `ifx_createMart()` stored procedure. If the column values of the parent table are not unique, the accelerator returns incorrect query results. This information can be set by using INDEX DISABLED keywords.

For example, by using flat files:

```
CREATE EXTERNAL TABLE "informix".nation SAMEAS nation_std
USING (DATAFILES("disk:/tmp/nation.tbl"),
      NUMROWS "num_rows");
CREATE UNIQUE INDEX nation_pk on nation (n_nationkey) disabled;
```

Or by using pipes:

```
CREATE EXTERNAL TABLE "informix".nation SAMEAS nation_std
USING (DATAFILES("pipe:/tmp/nation.pipe"),
      NUMROWS "num_rows");
CREATE UNIQUE INDEX nation_pk on nation (n_nationkey) disabled;
```



Important:

- The pipe must exist before you run the CREATE EXTERNAL TABLE statement.
- You must feed data into the pipe of the external table while the data is being loaded.

Where *num_rows* is the approximate number of rows that are contained in the external table. The query probing later uses the number of rows to determine which is the fact table of the probed queries. If you cannot provide *num_rows* in the external table definition, as an alternative, you can use the ENVIRONMENT Options clause of the SET OPTIMIZATION statement to define a general optimization environment for all queries in the current session. For example:

```
set environment use_dwa 'probe start';
set explain on avoid_execute;
SET OPTIMIZATION ENVIRONMENT FACT 'ext_d';
select * from ext_f, ext_d where ext_f.f1=ext_d.f1;
set environment use_dwa 'probe stop';
```

After the external tables are created, you can run query probing, create data marts, and load data.

After initially loading the data mart, you can use the `ifx_loadPartMart()` function to load more data from external tables into . The `ifx_loadPartMart()` function loads the complete data set, including recently inserted data.

- If your external table contains the complete table data set, you must replace the original data set with the more recent version. To replace the data set, run `ifx_dropPartMart()` and then run `ifx_loadPartMart()`. For example:

```
EXECUTE FUNCTION ifx_dropPartMart('demo_dwa','datamart_name','informix',
'nation', 'nation');
EXECUTE FUNCTION ifx_loadPartMart('demo_dwa','datamart_name','informix',
'nation', 'nation');
```

- If your external table contains only recently inserted data, run `ifx_loadPartMart()` to load the new data into the data mart alongside the existing data. For example:

```
EXECUTE FUNCTION ifx_loadPartMart('demo_dwa','datamart_name','informix',
'nation', 'nation');
```

Load data from views

You can use views to create a data mart and load it with data.

The columns contained in the views must consist of data types that are supported by , including JSON collections. For example, you can create a view that uses functions that are not supported natively by , if the columns that are output in the projection list are data types that are supported by .

Views are a projection of a SELECT statement, which can have joins.

The following functions are not supported for data marts that contain views:

- `ifx_refreshMart()`
- `ifx_setupTrickleFeed()`
- `ifx_dropPartMart()`
- `ifx_loadPartMart()`

If the parent table is a view, you must turn off the uniqueness checking on the **use_dwa** environment variable, and ensure uniqueness by other means.

Queries that are to be accelerated by using a data mart that contains views, must also use the views.

Related reference

[use_dwa session environment variable on page 130](#)

Load data from synonyms

You can use synonyms to create a data mart and load it with data.

A synonym is a name that you can use in place of another SQL table identifier.

The following functions are not supported for data marts that contain synonyms:

- ifx_refreshMart()
- ifx_setupTrickleFeed()
- ifx_dropPartMart()
- ifx_loadPartMart()

When you create a 1:n reference with a synonym as a parent table, you must ensure uniqueness. If the synonym references a local or remote table within the same Informix® instance, the uniqueness check is done automatically. If the synonym references a remote table on a different Informix® instance, you must turn off the uniqueness checking on the **use_dwa** environment variable, and ensure uniqueness by other means. If you create a 1:n reference with non-unique data values in the key columns of the parent table, you get incorrect query results for queries accelerated by Informix®.

If you have a query that contains a FACT directive with an alias defined for a synonym, you must use the alias name or the table name for the synonym with the FACT directive.

Query probing is not supported on queries that contain synonyms that were created on external tables of a remote database.

Related reference

[use_dwa session environment variable on page 130](#)

Load data from time series virtual tables

You can use time series virtual tables to create a data mart and load it with data.

Time series virtual tables are a layer in the Informix® server that you can use to present data in a different way, similar to a view. Time series virtual tables do not store the original data in a different format, but rather they provide access to the data as through it was stored in relational tables

Related information

[Data marts for time series data on page 118](#)

Example: in-memory acceleration of JSON

This example shows how you can use to accelerate data from JSON collections.

Before you begin

You must have installed and configured your MongoDB, Informix®, and environments.

1. From the MongoDB shell:
 - a. Create two collections and comments.

Example

```

$ mongo demo_database
MongoDB shell version: 2.4.9
connecting to: demo_database
mongos> db.comments.insert( [
  { uid:12345, pid:444, comment:"first" },
  { uid:12345, pid:888, comment:"second" },
  { uid:99999, pid:444, comment:"third" }
] )

```

- b. Create two users.

Example

```

mongos> db.users.insert( [
  { uid:12345, name:"john" },
  { uid:99999, name:"mia" }
] )
mongos> exit

```

2. From Informix® DB-Access utility:

- a. Create views on JSON collections.

Example

```

$dbaccess demo_database -
create view vcomments(uid,pid,comment) as select
  bson_value_int(data,'uid'),
  bson_value_int(data,'pid'),
  bson_value_varchar(data,'comment')
from comments;
create view vusers(uid,name) as select
  bson_value_int(data,'uid'),
  bson_value_varchar(data,'name')
from users;

set environment use_dwa 'probe cleanup';
set environment use_dwa 'probe start';
select {+ avoid_execute} * from vcomments c,vusers u where c.uid=u.uid;
set environment use_dwa 'probe stop';

```

- b. Create the data mart by using query probing.

Example

```

execute procedure ifx_probe2mart('demo_database','noSQL_mart');
execute function ifx_createmart('demo_dwa','noSQL_mart');
execute function ifx_loadmart('demo_dwa','noSQL_mart','NONE');

```

- c. Run the accelerated query.

Example

```

set environment use_dwa 'accelerate on';
select c.uid,name,comment from vcomments c,vusers u where c.uid=u.uid and
pid=444;

uid      12345
name     john
comment  first

```

```
uid      99999
name     mia
comment  third
```


Update a data mart

You can refresh the data in a data mart. You can update the data mart definition when you change the warehouse schema.

If you plan to use partial data refresh for data marts, including trickle feed, the value of the MAX_PDQPRIORITY configuration parameter must be set to 50 or greater during the initial data load and partial data refresh.

You can use different methods to update data marts depending on your needs. For best results, reload the entire data mart on a regular schedule, for example, once a week.

Table 10. Methods for refreshing data marts

Task	Method
Continuously refresh the data in the data mart	<p>Start trickle feed to refresh the data in the data mart as data is inserted in the database. You specify how frequently the data mart is refreshed.</p> <p>run the <code>ifx_setupTrickleFeed()</code> function.</p> <p> Important: You must periodically disable trickle feed and reload the data mart. Trickle feed stops running after 32000 refreshes.</p>
Refresh the data in the data mart when you choose	<p>Refresh the partitions in the data mart. Dimension and fact tables that have stale data are updated.</p> <p>run the <code>ifx_refreshMart()</code> function.</p>
Refresh a specific partition or table when you choose	<p>Drop the specified partition or table and then reload the data.</p> <p>Run the <code>ifx_dropPartMart()</code> function and then the <code>ifx_loadPartMart()</code> function.</p>
Reload all the data in the data mart	<p>Disable the data mart and then load the data. Any attached or detached fragments for dimension or fact tables are added or dropped. Query acceleration is temporarily disabled.</p>
Reload the data in your data marts once a week to ensure that they are accurate.	
Update the data mart definition by re-creating the data mart	<p>Drop the data mart and then re-create the data mart with the updated data mart definition. Query acceleration is temporarily disabled.</p>
Update the data mart definition by replacing the data mart	<p>Create a new data mart with an updated data mart definition, load the data, and then drop the original data mart. Query acceleration continues without interruption, however, you need enough main memory and disk space for two data marts.</p>

! **Important:** There is a 32 KB limit on the number of refreshes from Informix® to Informix® Warehouse Accelerator before a reload is required. If you exceed the limit, an alert_type ERROR with alert_color YELLOW is added to the **sysadmin:ph_alert** table indicating the limit is reached. Here are some examples of trickle feed frequency and the number of days before reload is required:

Frequency	Number of days before reload
5 minutes	113 days
15 minutes	341 days
60 minutes	1365 days

Related reference

[ifx_dropPartMart\(\) function on page 140](#)

[ifx_loadPartMart\(\) function on page 151](#)

[ifx_refreshMart\(\) function on page 154](#)

[ifx_setupTrickleFeed\(\) function on page 159](#)

Reloading the data mart

You can reload all the data in the data mart. Query acceleration is temporarily suspended.

About this task

You do not need to stop query acceleration. Between the time when you set the data mart to disabled mode and load the data mart, query acceleration is suspended and all queries are run by the Informix® database server.

To reload the data mart:

1. If trickle feed is enabled, disable it.
2. Put the data mart in `Disabled` state.
Query acceleration is suspended.
3. Load the data mart.
When the load completes, query acceleration resumes.
4. **Optional:** Enable trickle feed for the data mart.

Example

The following statements disable trickle feed for the data mart, put the data mart in `Disabled` state, load the data mart, and enable trickle feed for the data mart:

```
EXECUTE FUNCTION ifx_removeTrickleFeed('MyAccelerator', 'Datamart1');
```

```
EXECUTE FUNCTION ifx_setMart('MyAccelerator', 'Datamart1' 'OFF');

EXECUTE FUNCTION ifx_loadMart('MyAccelerator', 'Datamart1', 'MART');

EXECUTE FUNCTION ifx_setupTrickleFeed('MyAccelerator', 'Datamart1', 5);
```

Re-creating a data mart

You can re-create a data mart to update the data mart definition. You can add or remove dimension and fact tables to or from the data mart. You drop the data mart and re-create it with the new definition. Query acceleration is temporarily suspended.

About this task

Update the data mart definition when you make the following types of schema modifications to the data warehouse:

- Rename tables
- Rename columns
- Alter tables to add or drop columns
- Change the data types of the columns
- Add or drop constraints

You do not need to disable query acceleration. Between the time when you drop and load the data mart, query acceleration is suspended and all queries are run by the Informix® database server.

To re-create a data mart:

1. **Optional:** Create a data mart definition manually.
2. If trickle feed is enabled, disable it.
3. Drop the data mart.
Query acceleration is suspended.
4. Create the data mart.
You can use the same data mart name as the original data mart.
5. Load the data mart.
When the load completes, query acceleration resumes.
6. Enable trickle feed for the data mart.

Example

This example shows how to re-create a data mart after the new data mart definition is stored in the **stores_marts** database. The following statements disable trickle feed for the data mart, drop the data mart, re-create the data mart using the new data mart definition, load the data mart, and enable trickle feed for the data mart:

```
EXECUTE FUNCTION ifx_removeTrickleFeed('MyAccelerator', 'customermart1');

EXECUTE FUNCTION ifx_dropMart('MyAccelerator', 'customermart1');
```

```
EXECUTE FUNCTION ifx_createMart('MyAccelerator', 'customermart1', 'stores_marts');
EXECUTE FUNCTION ifx_loadMart(('MyAccelerator', 'customermart1', 'MART');
EXECUTE FUNCTION ifx_setupTrickleFeed('MyAccelerator', 'customermart1', 5);
```

Related information

[Creating data mart definitions manually by using workload analysis on page 101](#)

Replacing a data mart

You can replace a data mart to update the data mart definition. You create a new data mart with a new definition and drop the original data mart. Query acceleration continues uninterrupted.

Before you begin

You need enough disk space to contain both data marts until you drop the original data mart.

About this task

Update the data mart definition when you make the following types of schema modifications to the data warehouse:

- Rename tables
- Rename columns
- Alter tables to add or drop columns
- Change the data types of the columns
- Add or drop constraints

Query acceleration runs on the original data mart until the new data mart is loaded. As soon as the new data mart is loaded, the database server uses the new data mart for matching new queries to AQTs.

To replace a data mart:

1. **Optional:** Create a data mart definition manually.
2. Create the new data mart.
3. Load the new data mart.
 - Candidate queries that are sent to the accelerator server use the new data mart.
4. Enable trickle feed for the new data mart.
5. If trickle feed is enabled for the original data mart, disable it.
6. Drop the original data mart.

Example

This example shows how to replace a data mart after the new data mart definition is stored in the **stores_marts** database. The original data mart is named **customermart1** and the new data mart is named named **customermart2**. The following

statements create the new data mart, load the new data mart, enable trickle feed for the new data mart, disable trickle feed for the original data mart, and drop the original data mart:

```
EXECUTE FUNCTION ifx_createMart('MyAccelerator', 'customermart2', 'stores_marts');  
  
EXECUTE FUNCTION ifx_loadMart(('MyAccelerator', 'customermart2', 'MART');  
  
EXECUTE FUNCTION ifx_setupTrickleFeed('MyAccelerator', 'customermart2', 5);  
  
EXECUTE FUNCTION ifx_removeTrickleFeed('MyAccelerator', 'customermart1');  
  
EXECUTE FUNCTION ifx_dropMart('MyAccelerator', 'customermart1');
```

Related information

[Creating data mart definitions manually by using workload analysis on page 101](#)

Drop a data mart

You can drop a data mart if you no longer need the data mart or if you need to update the data in a data mart.

Drop a data mart by using one of the following tools:

- The SQL administration [ifx_dropMart\(\) function on page 139](#).
- Java™ classes.

Related information

[Java classes for Informix Warehouse Accelerator on page 60](#)

Data marts for time series data

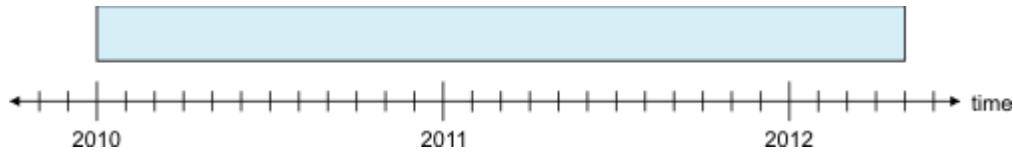
You can create data marts that contain tables that represent time series data. Typically, such tables are fact tables in the data mart.

The **TimeSeries** data type is not directly supported by , but you can create a data mart for time series data based on a virtual table. The virtual table is a relational representation of a table that has a **TimeSeries** column. Use the virtual table in your data mart definition, not the table that contains the **TimeSeries** column.

Before you can create a data mart for time series data, you must create a virtual table that is based on the time series table, by using the TSCreateVirtualTab procedure. The TSVTMode parameter of the TSCreateVirtualTab procedure must include both flags **scan_discreet** and **scan_regular_discreet** setting.

When you load the data mart on the accelerator, the time series data is retrieved from the virtual table. There is no in-memory copy of the data.

By default, the time series data on the accelerator is not partitioned.



Unpartitioned data can slow the load speed and query execution time, and to refresh the data you must reload the complete virtual table. However, you can virtually partition the time series data to limit the amount of data on the accelerator. Virtual partitions can be refreshed without reloading the complete data set.

Related information

[Refresh time series virtual partition on page 122](#)

[Load data from time series virtual tables on page 112](#)

Define virtual partitions by assigning a calendar

To virtually partition time series data, a time series calendar must be assigned to the time series virtual table in a data mart. Use the `ifx_TSDW_setCalendar()` routine to assign a calendar to a time series virtual table in a specific data mart.

The calendar definition includes a start time stamp and an interval size. Time series data in the virtual table with a time stamp earlier than the calendar start time stamp is not included in the data mart. The calendar interval size defines the size of the virtual partition.

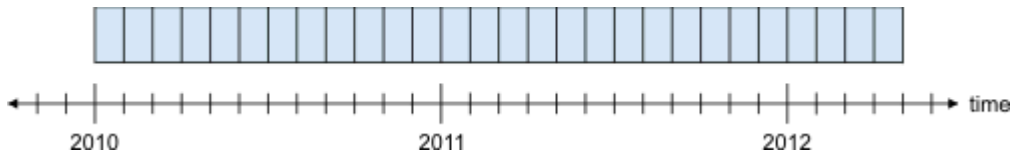
You can create a calendar or use an existing calendar, for example a built-in calendar. Typically, you use a calendar for virtual partitions that has a significantly longer interval than the calendar for the time series. For example, you can create a calendar where the start time stamp is January 2010 and the interval size is one month. You can assign it to a time series virtual table, by running the following statements:

```
insert into calendartable (c_name, c_calendar) values
('2010monthly', 'startdate(2010-01-01 00:00:00.00000),
pattstart(2010-01-01 00:00:00.00000), pattern({1 on},month)');
execute function ifx_TSDW_setCalendar('demo_dwa', 'demo_mart',
'informix', 'ts_data_v', '2010monthly');
```



Important: You must assign a calendar to a time series virtual table before the data mart is initially loaded (the data mart must be in LoadPending state). To specify a different calendar after the data mart is initially loaded, you must drop and recreate the data mart.

After a calendar is assigned to the time series virtual table and the data mart has been loaded, the time series data on the accelerator is arranged in virtual partitions:



Related reference

[ifx_TSDW_setCalendar\(\) function on page 168](#)

Identify a virtual partition

A virtual partition is identified by a calendar index or a time stamp.

The calendar index is a positive integer value, where 0 is the first virtual partition. For example, for the 2010monthly calendar used in the example, the calendar index 0 identifies the virtual partition for January 2010, and the calendar index 1 identifies February 2010.

When using a time stamp, the time stamp value is internally mapped into a calendar index. For the 2010monthly calendar used in the example, all time stamps between `'2010-01-01 00:00:00.00000'::datetime year to fraction(5)`, including this value, and `'2010-02-01 00:00:00.00000'::datetime year to fraction(5)`, excluding this value, identify the virtual partition for January 2010.

The time stamp is automatically extended to `'datetime year to fraction(5)'`. For example, `'2010-01'::datetime year to month` is equivalent to `'2010-01-01 00:00:00.00000'::datetime year to fraction(5)'`.

Limit the size of data with time windows

You can limit the amount of data in a data mart for time series data by using time windows.

Create time windows

After you assign a calendar, you can create time windows by using the `ifx_TSDW_createWindow()` routine. A time window consists of one or more adjacent virtual partitions. You can create multiple time windows. Time windows cannot overlap each other.



Note: If you create time windows, the time series data remains partitioned in the time windows until the data mart is dropped. To load all time series data into the data mart without the time windows, you must drop the data mart and re-create without the time windows.

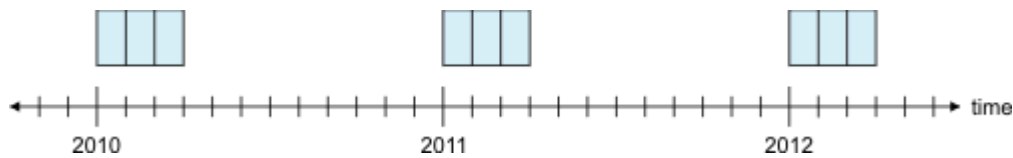
A time window is identified by a begin virtual partition and an end virtual partition. The begin virtual partition is the first virtual partition that is included in the time window, while the end virtual partition is the first virtual partition that is excluded from the time window. For example, if you want to compare the data of the first 3 months of the year for 2010, 2011, and 2012, you can create three time windows: January through March for each year, each of which contains three virtual partitions, by running the following statements:

```
ifx_TSDW_createWindow('demo_dwa', 'demo_mart', 'informix', 'ts_data_v', 0, 3);
ifx_TSDW_createWindow('demo_dwa', 'demo_mart', 'informix', 'ts_data_v', 12, 15);
ifx_TSDW_createWindow('demo_dwa', 'demo_mart', 'informix', 'ts_data_v', 24, 27);
```

Or by using time stamps to identify the virtual partitions:

```
ifx_TSDW_createWindow('demo_dwa', 'demo_mart', 'informix', 'ts_data_v',
'2010-01'::datetime year to month, '2010-04'::datetime year to month);
ifx_TSDW_createWindow('demo_dwa', 'demo_mart', 'informix', 'ts_data_v',
'2011-01'::datetime year to month, '2011-04'::datetime year to month);
ifx_TSDW_createWindow('demo_dwa', 'demo_mart', 'informix', 'ts_data_v',
'2012-01'::datetime year to month, '2012-04'::datetime year to month);
```

This figure shows three time windows that each contain three virtual partitions:



The initial window for a time series virtual table must be created before the data mart is initially loaded (the data mart must be in LoadPending state). Subsequent time windows can be created either before the data mart is initially loaded, or when the data mart is in Active state. When you create a time window in an Active data mart, the time series data for the new time window is loaded to the data mart immediately.

Remove time windows

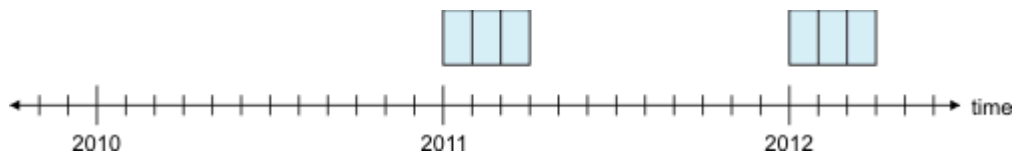
To remove a time window and the time series data it contains from the accelerator, use the `ifx_TSDW_dropWindow()` routine. The time window to be dropped is identified by the virtual partition it starts with. For example, if you want to remove the time window for January to March 2010, run the following statement:

```
ifx_TSDW_dropWindow('demo_dwa', 'demo_mart', 'informix', 'ts_data_v', 0);
```

Or by using a time stamp to identify the time window:

```
ifx_TSDW_dropWindow('demo_dwa', 'demo_mart', 'informix', 'ts_data_v',
'2010-01'::datetime year to month);
```

This figure shows two time windows that each contain three virtual partitions.



You can remove time windows when the data mart is in LoadPending or Active state. When you remove a time window in an Active data mart, the time series data for the time window is removed from the data mart immediately.

Move time windows

After you create time windows and load the data into the data mart, you can move the time windows forward or backward by a specified number of virtual partitions. When you move a time window, any virtual partition that is no longer contained in the time window is dropped from the data mart. Any virtual partition that is included in the moved time window is loaded into the data mart. You move time windows by running the `ifx_TSDW_moveWindows()` routine. The virtual partition default value is 1.

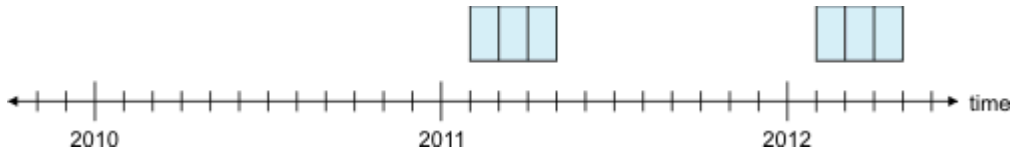
For example, you can move the time windows so that you can query on the second 3 months of the years 2011 and 2012 by running the following statement:

```
ifx_TSDW_moveWindows('demo_dwa', 'demo_mart', 'informix', 'ts_data_v');
```

That is equivalent to:

```
ifx_TSDW_moveWindows('demo_dwa', 'demo_mart', 'informix', 'ts_data_v', 1);
```

This figure shows that two time windows are moved forward in time by one time interval:



Related reference

[ifx_TSDW_createWindow\(\) function on page 162](#)

[ifx_TSDW_dropWindow\(\) function on page 163](#)

[ifx_TSDW_moveWindows\(\) function on page 165](#)

[ifx_TSDW_updatePartition\(\) function on page 166](#)

Refresh time series virtual partition

After you define virtual partitions and load the data mart, you can refresh the time series data for a single virtual partition by running the `ifx_TSDW_updatePartition()` routine.

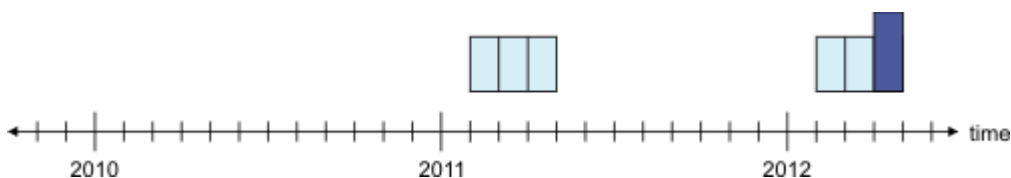
For example, you can refresh the time series data for March 2012 by running the following statement:

```
ifx_TSDW_updatePartition('demo_dwa', 'datamart_name', 'informix', 'ts_data_v', 26);
```

or by using a time stamp to identify the virtual partition:

```
ifx_TSDW_updatePartition('demo_dwa', 'demo_mart', 'informix', 'ts_data_v',  
'2012-03'::datetime year to month);
```

This figure shows that the last month in the 2012 time period is updated:



Related information

[Data marts for time series data on page 118](#)

Example: Accelerating time series data

Use this step-by-step example as a guide to accelerate Informix® time series data by using .

The SQL statements that are used in this example are run in the DB-Access utility, and are preceded by `>`. Commands that are run from the shell are preceded by `$`.

This example uses the **stores_demo** database that is created by the command `dbaccessdemo`. To create the **stores_demo** database, run the following command:

```
$ dbaccessdemo stores_demo -log
```

The workload in this example consists of the following two queries:

```
select min(tstamp), max(tstamp) from ts_data_v;
select first 3 state,avg(value) average
  from ts_data_v v, Customer_ts_data l, customer c
  where v.loc_esi_id = l.loc_esi_id and l.customer_num = c.customer_num
  group by 1 order by 2 desc;
```

The first query selects the minimum and maximum time stamp from the `ts_data_v` table. The second query selects the top three states and the average of the `'value'` column.

Step 1: Connect to database stores_demo

```
$ dbaccess - -
> database stores_demo;
Database selected.
```

Step 2: Run and analyze workload on Informix®

```
> set environment use_dwa 'probe cleanup';
Environment set.
> set environment use_dwa 'probe start';
Environment set.
> select min(tstamp), max(tstamp) from ts_data_v;
(min)                (max)
2012-11-10 00:00:00.00000 2011-02-07 23:45:00.00000
1 row(s) retrieved.
> select first 3 state,avg(value) average
  from ts_data_v v, Customer_ts_data l, customer c
  where v.loc_esi_id = l.loc_esi_id and l.customer_num = c.customer_num
  group by 1 order by 2 desc;
state                average
DE      0.70850671296296
NY      0.47548564814815
FL      0.42491203703704
3 row(s) retrieved.
> set environment use_dwa 'probe stop';
Environment set.
```

Step 3: Convert results of workload analysis to data mart definition, create and load the data mart

```
> execute procedure ifx_probe2Mart('stores_demo','demo_mart');
Routine executed.
> execute function ifx_createMart('demo_dwa','demo_mart');
(expression) The operation was completed successfully.
1 row(s) retrieved.
```

```
> execute function ifx_loadMart('demo_dwa','demo_mart','NONE');
(expression) The operation was completed successfully.
1 row(s) retrieved.
```

Step 4: Run workload on

```
> set environment use_dwa 'accelerate on';
Environment set.
> select min(tstamp), max(tstamp) from ts_data_v;
(min)                                (max)
2012-11-10 00:00:00.00000 2011-02-07 23:45:00.00000
1 row(s) retrieved.
> select first 3 state,avg(value) average
   from ts_data_v v, Customer_ts_data l, customer c
   where v.loc_esid = l.loc_esid and l.customer_num = c.customer_num
   group by 1 order by 2 desc;
state          average
DE      0.70850671296296
NY      0.47548564814814
FL      0.42491203703704
3 row(s) retrieved.
```

Step 5: Drop the data mart

```
> execute function ifx_dropMart('demo_dwa','demo_mart');
(expression) The operation was completed successfully.
1 row(s) retrieved.
```

Step 6: Create the data mart again

```
> execute function ifx_createMart('demo_dwa','demo_mart');
(expression) The operation was completed successfully.
1 row(s) retrieved.
```

Step 7: Assign calendar 'ts_1month' to table 'ts_data_v'

```
> execute function ifx_TSDW_setCalendar(
  'demo_dwa','demo_mart','informix','ts_data_v','ts_1month');
(expression) The operation was completed successfully.
1 row(s) retrieved.
```

Step 8: Create time window for January 2011

```
> execute function ifx_TSDW_createWindow(
  'demo_dwa','demo_mart','informix','ts_data_v',
  '2011-01'::datetime year to month, '2011-02'::datetime year to month);
(expression) The operation was completed successfully.
1 row(s) retrieved.
```

Step 9: Load the data mart

```
> execute function ifx_loadMart('demo_dwa','demo_mart','NONE');
(expression) The operation was completed successfully.
1 row(s) retrieved.
```

Step 10: Run workload on

```
> select min(tstamp), max(tstamp) from ts_data_v;
(min)                (max)
2011-01-01 00:00:00.00000 2011-01-31 23:45:00.00000
1 row(s) retrieved.
> select first 3 state,avg(value) average
   from ts_data_v v, Customer_ts_data l, customer c
   where v.loc_esid = l.loc_esid and l.customer_num = c.customer_num
   group by 1 order by 2 desc;
state                average
DE      1.36793313172043
NY      0.51537264784946
OK      0.51153326612903
3 row(s) retrieved.
```

Step 11: Move time window forward one month from January to February

```
> execute function ifx_TSDW_moveWindows(
'demo_dwa','demo_mart','informix','ts_data_v');
(expression) The operation was completed successfully.
1 row(s) retrieved.
```

Step 12: Run workload on

```
> select min(tstamp), max(tstamp) from ts_data_v;
(min)                (max)
2011-02-01 00:00:00.00000 2011-02-07 23:45:00.00000
1 row(s) retrieved.
> select first 3 state,avg(value) average
   from ts_data_v v, Customer_ts_data l, customer c
   where v.loc_esid = l.loc_esid and l.customer_num = c.customer_num
   group by 1 order by 2 desc;
state                average
MA      0.92973660714286
NY      0.46185416666667
NJ      0.38778348214286
3 row(s) retrieved.
```

Step 13: Add a new measurement reading for a customer in NY

```
> insert into ts_data_v(loc_esid,measure_unit,direction,tstamp,value) values
('4727354321757220','KWH','P',
'2011-02-08 00:15'::datetime year to minute, 500);
1 row(s) inserted.
```

Step 14: Update the virtual partition of February 2011

```
> execute function ifx_TSDW_updatePartition(
'demo_dwa','demo_mart','informix','ts_data_v',
'2011-02'::datetime year to month);
(expression) The operation was completed successfully.
1 row(s) retrieved.
```

Step 15: Run workload on

```
> select min(tstamp), max(tstamp) from ts_data_v;
(min)                (max)
```

```

2011-02-01 00:00:00.000000 2011-02-08 00:15:00.000000
1 row(s) retrieved.
> select first 3 state,avg(value) average
   from ts_data_v v, Customer_ts_data l, customer c
   where v.loc_esid = l.loc_esid and l.customer_num = c.customer_num
   group by 1 order by 2 desc;
state          average
NY             1.20410995542348
MA             0.92973660714286
NJ             0.38778348214286
3 row(s) retrieved.

```

Step 16: Remove time window for February 2011

```

> execute function ifx_TSDW_dropWindow(
'demo_dwa','demo_mart','informix','ts_data_v',
'2011-02'::datetime year to month);
(expression) The operation was completed successfully.
1 row(s) retrieved.

```

Step 17: Run workload on

```

> select min(tstamp), max(tstamp) from ts_data_v;
(min)                (max)

1 row(s) retrieved.
> select first 3 state,avg(value) average
   from ts_data_v v, Customer_ts_data l, customer c
   where v.loc_esid = l.loc_esid and l.customer_num = c.customer_num
   group by 1 order by 2 desc;
state          average
No rows found.

```

Step 18: Drop the data mart

```

> execute function ifx_dropMart('demo_dwa','demo_mart');
(expression) The operation was completed successfully.
1 row(s) retrieved.
Database closed.

```

Example: Continuous refresh of accelerated time series data

Use this step-by-step example as a guide to continuously refresh accelerated Informix® time series data by using .

The example assumes that you have already setup and configured and that you have a working data mart.

Step 1: Define the continuous refresh procedure

Insert a row into the scheduler in the **sysadmin** database to add the continuous refresh stored procedure. The accelerator name and data mart name are provided:

```

create dba function
informix.ifx_continuous_refresh
(task_id integer, task_seq integer, accel varchar(128), mart varchar(128))
returning integer

```



```

define task_interval like sysadmin:ph_task.tk_frequency;
define mart_id       like sysadmin:iwa_datamarts.m_mart_id;
define mart_dbname   like sysadmin:iwa_datamarts.m_dbname;
define table_owner   like sysadmin:iwa_marttables.mt_owner;
define table_name     like sysadmin:iwa_marttables.mt_tabname;
define table_id      like sysadmin:iwa_marttables.mt_tab_id;
define cal_name       like sysadmin:iwa_tsvt_partcal.calname;
define has_windows    like sysadmin:iwa_tsvt_partcal.partitioned;
define cal_startdate  datetime year to fraction(5);
define upd_begin      integer;
define upd_end        integer;
define win_begin      integer;
define win_end        integer;
define index          integer;
define upd_result     lvarchar;
define sql_err        integer;

on exception set sql_err
    return sql_err;
end exception

```

Step 2: Define the frequency of refresh

Define the frequency of the refresh.

```

select tk_frequency into task_interval
    from sysadmin:ph_task where tk_id = task_id;

```

Step 3: Add the data mart definition from the sysadmin database

Add the data mart definition from the **sysadmin** database. The data mart must already exist.

```

-- Get the data mart information from the sysadmin database.
select m_mart_id, m_dbname
    into mart_id, mart_dbname from sysadmin:iwa_datamarts
    where m_accel_name=accel and m_name=mart;

-- The data mart must already exist.
if mart_id is NULL then return -1; end if
-- The data mart must be defined in the current database.
if mart_dbname != DBINFO('dbname') THEN return -2; end if

-- For each fact table that is a time series virtual table.
foreach select mt_tab_id, mt_owner, mt_tabname
    into table_id, informix, table_name
    from sysadmin:iwa_marttables mt,
        informix.systables t, informix.sysams a
    where mt_mart_id = mart_id
        and mt.mt_isfact = 1
        and mt.mt_tabid = t.tabid
        and a.am_id = t.am_id
        and a.am_name = 'ts_vtam'

```

Step 4: Add the time series virtual table partitioning calendar

Add the time series virtual table from the **sysadmin** database. If there is no partitioning calendar, the virtual table is not included.

```

-- Get the time series virtual table partitioning calendar.
select calname, parted into cal_name, has_windows
  from sysadmin:iwa_tsvt_partcal
  where tab_id = table_id;

-- If no partitioning calendar, skip the time series virtual table.
if cal_name is null then continue foreach; end if

```

Step 5: Define the calendar

Define the calendar by querying the time series table with the **CalStartDate** function to obtain the existing start date, and then refresh the time series data with the **ifx_TSDW_updatePartition()** function to get the latest information.

```

-- Get the calendar start date.
execute function CalStartDate(cal_name) into cal_startdate;
if cal_startdate is NULL then continue foreach; end if

-- Get the calendar index of current time and update the end index.
execute function CalIndex(
  cal_name, cal_startdate, current) into upd_end;
if upd_end is NULL or upd_end < 0 then continue foreach; end if
let upd_end = upd_end + 1;

-- Get the calendar index of current time and update the begin index.
execute function CalIndex(
  cal_name, cal_startdate, current-task_interval) into upd_begin;
-- An index value less than zero is not allowed.
if upd_begin < 0 then let upd_begin = 0; end if

-- If the time windows are defined:
if has_windows = 1 then
  -- Get all time windows that overlap the calendar start and end times.
  foreach select begin, end into win_begin, win_end
    from sysadmin:iwa_tsvt_windows
    where tab_id = table_id
    and end > upd_begin and begin < upd_end

    if win_begin < upd_begin then let win_begin = upd_begin; end if
    if win_end > upd_end then let win_end = upd_end; end if

    let index = win_begin;
    while index < win_end
      execute function ifx_TSDW_updatePartition(
        demo_dwa, datamart_name, informix, ts_data_v,
        calendar_index)
        into upd_result;
      let index = index + 1;
    end while
  end foreach
else
  -- If time windows are not defined.
  let index = upd_begin;
  while index < upd_end
    execute function ifx_TSDW_updatePartition(
      demo_dwa, datamart_name, informix, ts_data_v, calendar_index)
      into upd_result;
    let index = index + 1;
  end while

```

```

        end if
    end foreach

    return 0;

end function;

```

Step 6: Grant access to the scheduler

Use the GRANT statement to give access to the continuous refresh stored procedure in the **sysadmin** database scheduler.

```

grant execute on function
    informix.ifx_continuous_refresh(integer,integer,vvarchar,vvarchar)
to public as informix;

```

Step 7: Add the continuous refresh procedure to the scheduler

Add an entry to the scheduler **ph_task** table in the **sysadmin** database to refresh the data every 15 minutes.

```

insert into 'sysadmin':ph_task
    (tk_name,
tk_description,
tk_type,
tk_dbs,      -- The database where the data mart is located.
tk_execute, -- The continuous refresh procedure to run.
tk_delete,
tk_start_time, -- Whether to start immediately or later.
tk_stop_time, -- The stop time or never stop (NULL).
tk_frequency, -- Indicates the frequency, which in this case is 15 minutes.
tk_enable)
values (
    'ifx_continuous_refresh_myAccel_myMart',
    'trickle feed for data mart myMart@myAccel',
    'TASK',
    'myDatabase',
    'execute function ifx_continuous_refresh($DATA_TASK_ID,$DATA_SEQ_ID,
        "demo_dwa","demo_mart")',
    NULL,
    datetime(00:00:00) hour to second,
    NULL,
    interval(15) minute to minute, -- The frequency in which the task is run.
    't');

```

Turning on query acceleration

Before queries can be routed to the accelerator server for processing, you must set the `use_dwa` session environment variable.

About this task

Turn on acceleration by enabling the `use_dwa` session environment variable:

```

SET ENVIRONMENT USE_DWA 'accelerate on';

```

Setting the `use_dwa 'accelerate on'` option of the SET ENVIRONMENT statement of SQL enables the Informix® query optimizer to consider using the accelerator server to process the query when the optimizer generates the query plans.

Table 11. Keywords and variables in the use_dwa environment variable

Keyword and variable	Purpose	Restrictions
accelerate	Controls query acceleration. By default, queries are not accelerated.	
accelerate on	Turns on acceleration. Queries that match one of the accelerated query tables (AQTs) are sent to the accelerator server for processing. Use this setting to accelerate queries.	
accelerate off	Turns off acceleration. Queries are not sent to the accelerator server even if the queries meet the required criteria.	
fallback	Controls what happens when cannot accelerate the queries. For example, the accelerator server is offline or the queries do not match one of the accelerated query tables (AQTs). The fallback on option is the default setting.	The fallback option applies only if you set the accelerate on option.
fallback off	If cannot accelerate the queries, the queries are not sent to the Informix® database server for processing. Use the fallback off option when you do not want subsequent queries that are processed by the Informix® database server.	
fallback on	If cannot accelerate the queries, the queries are sent to the Informix® database server for processing.	
probe	Activates query probing. Query probing is gathering information about a query workload. Query probing is used in workload analysis to create a data mart definition. The probe stop option is the default setting.	


Table 11. Keywords and variables in the use_dwa environment variable

(continued)

Keyword	Purpose	Restrictions
probe start	Activates query probing.	
probe stop	Deactivates query probing.	
debug	Activates debugging. The debug off option is the default setting.	The debug option applies only if you set the accelerate on option or the probe start option.
debug on	Turns on debugging. The default log file is the <code>online.log</code> file as specified by the MSGPATH configuration parameter.	
debug off	Turns off debugging.	
debug file	Sets a debug log file. Use the <code>file_name</code> option to specify a different log file than the default <code>online.log</code> file. The default log file is specified by the MSGPATH configuration parameter.	The debug on option is required for the debug file option to take effect. If you specify a file name, use a path and file name that does not have any special characters, such as new lines, spaces, or tabs. The value of <code>file_name</code> is case-sensitive. If you later specify the debug file option without the <code>file_name</code> , the debugging output destination resets to the default <code>online.log</code> file as specified by the MSGPATH configuration parameter.
uniquecheck on	This parameter controls uniqueness checking during the creation of a data mart. The uniquecheck on is the default setting.	
uniquecheck off	You can use uniquecheck off to skip checking for the relevant index of parent table when creating a data mart. With uniquecheck off , you can create 1:n references where the parent table is a synonym that references a remote table or a view.	If you specify uniquecheck off , you must ensure the uniqueness of data in the parents key columns by other means. If you create a 1:n reference with non-unique data values in the key columns of the parent table, you will get incorrect query results for queries accelerated by Informix®.

Table 11. Keywords and variables in the use_dwa environment variable

(continued)

Keyword	Purpose	Restrictions
session	Sets the use_dwa environment variable settings for a different session as identified by the <i>session_id</i> number.	Only user informix can specify the session option. The SESSION ID must be numeric and must identify an existing session.
session_id	Use this option to change the settings for a session that originates from an application that opens the connection to the database one time and does not close the connection.	 Important: Do not include the session session_id option in the sysdbopen() stored procedure.
loadpdq	This parameter controls the PDQPRIORITY setting for the database sessions that extract data from the database tables during a data mart load operation.	<i>resources</i> is a positive integer value between 0 and 100 (inclusive).
resources	The setting is effective for full data mart load, partition load and refresh mart. As with other database sessions, this PDQPRIORITY is then used together with the MAX_PDQPRIORITY configuration parameter to calculate the effective PDQ priority for the data extraction database sessions. The default PDQPRIORITY for data extraction database sessions is 2.	
probe cleanup	The probe cleanup option removes all probing data that is produced by all past and current sessions in the database.	

Usage

The SET ENVIRONMENT use_dwa statement takes one argument, which is a string value within single or double quotation marks. Keywords are not case-sensitive.

Example

Examples

The following example turns on acceleration.

```
set environment use_dwa 'accelerate on';
```

The following example removes all the probing data that was previously collected for the current database.

```
set environment use_dwa 'probe cleanup';
```

The following example turns on acceleration and turns off fallback. The queries are not sent to the Informix® database server for processing. Queries that cannot be accelerated by will fail.

```
set environment use_dwa 'accelerate on';
set environment use_dwa 'fallback off';
```

The following example sets the debug option. By default, the debug information is appended to the `online.log` file.

```
set environment use_dwa 'debug on';
```

The following example creates probing data for your queries, turns on debugging, and appends the debugging information to a file named `/tmp/my_debug_file`.

```
set environment use_dwa 'probe start';
set environment use_dwa 'debug on';
set environment use_dwa 'debug file /tmp/my_debug_file';
```

The following example turns on debugging. The debug output of query 1 is written to the file `/tmp/myDwaDebugFile`. The debug output of query 2 is written to the default `online.log` file.

```
set environment use_dwa 'debug on';
set environment use_dwa 'debug file /tmp/myDwaDebugFile';
select ... { query 1 }
set environment use_dwa 'debug file';
select ... { query 2 }
```

The following text shows example debug output. This output shows that a query has matched an AQT and is being sent to for processing.

```
15:00:35 SQDWA: select MIN(s_suppkey) from partsupp x0 left join supplier x1
on x0.ps_
15:00:35 SQDWA: Identified 1 candidate AQTs for matching
15:00:35 SQDWA: matched: aqt48a93f10-d192-404f-8911-e01d67aadde2
15:00:35 SQDWA: matching successful (0 msec) aqt48a93f10-d192-404f-8911-
e01d67aadde2
15:00:35 SQDWA: offloading successful (22 msec)
```


SQL administration routines

You can use the functions and procedures from the command line or in an application to automate administration tasks.

The names of the functions and the procedures are not case sensitive.

Here are the character limits for the SQL administration routine elements:

- Accelerator name length - 128 characters
- IP address length - 256 characters
- Data mart name length - 128 characters
- Schema name length - 128 characters
- Table name length - 128 characters
- Column name length - 128 characters
- Authentication token length - 100 characters (generated by the system)

You must have permissions to administer . For more information, see [Permissions for administering Informix Warehouse Accelerator on page 59](#).

You cannot run functions and procedures from the **sysadmin** database.

Related information

[Permissions for administering Informix Warehouse Accelerator on page 59](#)

SQL administration routines sorted by task

SQL administration routines are sorted into logical areas based on the type of task.

Table 12. SQL administration routines by task type

Task type	Description
SQL routines for administering accelerators	Create an accelerator in an accelerator server: ifx_setupDWA() function on page 158 Remove (drop) an accelerator from the accelerator server: ifx_removeDWA() function on page 156
SQL routines for gathering accelerator server information	Return metrics about an accelerator: ifx_getdwaMetrics() function on page 142
SQL routines for administering data marts	Create a data mart in an accelerator: ifx_createMart() function on page 136 Populate an empty data mart in an accelerator with data: ifx_loadMart() function on page 149 Load the data into one partition in a fragmented table or into a non-fragmented table: ifx_loadPartMart() function on page 151 Drop (remove) the data mart from an accelerator: ifx_dropMart() function on page 139

Table 12. SQL administration routines by task type (continued)

Task type	Description
	<p>Remove the data in one partition in a fragmented table or all of the data from a non-fragmented table: ifx_dropPartMart() function on page 140</p> <p>Change the state of a data mart from Active to Disabled or from Disabled to Active: ifx_setMart() function on page 157</p> <p>Check for data changes in individual partitions and refreshes the changed partitions: ifx_refreshMart() function on page 154</p> <p>Enable the continuous refreshing of data in the fact and dimension tables of the data mart as the data changes in the database: ifx_setupTrickleFeed() function on page 159</p> <p>Stop the trickle feed so that data in the data mart is no longer continuously refreshed: ifx_removeTrickleFeed() function on page 156</p>
SQL routines for gathering data mart information	<p>Return a set of unnamed rows that contain the status for all the data marts in an accelerator: ifx_listMarts() function on page 148</p> <p>Return the status of a single data mart as an unnamed row: ifx_getMartStat() function on page 146</p> <p>Retrieve the data mart definition from an accelerator and return it as a character large object (CLOB) in XML format: ifx_getMartdef() function on page 145</p>
SQL routines for administering data mart schemas	<p>Convert the data that is gathered from probing into a data mart definition: ifx_probe2Mart() procedure on page 152</p> <p>Return a character large object (CLOB) that contains the data mart definition in XML format: ifx_genMartDef() function on page 141</p> <p>Convert a data mart definitions from XML format into data mart schema tables: ifx_xml2Mart() procedure on page 169</p> <p>Use the ifx_getMartdef() function to retrieve the data mart definition from an accelerator; and then use the ifx_xml2Mart() procedure to store the data mart definition XML file in the data mart schema tables: ifx_def2Mart() procedure on page 138</p>
SQL routines for administering time series virtual tables	<p>Assign a TimeSeries calendar: ifx_TSDW_setCalendar() function on page 168</p> <p>Create a time window: ifx_TSDW_createWindow() function on page 162</p> <p>Remove a time window: ifx_TSDW_dropWindow() function on page 163</p> <p>Move time windows: ifx_TSDW_moveWindows() function on page 165</p> <p>Refresh a virtual partition: ifx_TSDW_updatePartition() function on page 166</p>

ifx_createMart() function

The ifx_createMart() function creates a data mart in an accelerator.

There are two types of ifx_createMart() functions and each type takes different parameters.

- A type that creates a data mart that is based on a data mart definition that is in XML format. You create the data mart definition by using workload analysis. The ifx_createMart() function retrieves the data mart definition as a CLOB data type.
- A type that creates a data mart that is based on the schema of the database tables that you create with the ifx_probe2Mart() procedure.

Syntax

- Here is the syntax that is used to create a data mart that is based on a data mart definition.

```
ifx_createMart('accelerator_name','data_mart_definition')
```

accelerator_name

The name of the accelerator where you want to create the data mart.

data_mart_definition

The data mart definition. The name of the data mart is retrieved from `mart_name` attribute of the data mart definition. Valid data mart names have a maximum of 128 characters. Data marts names must be unique in an accelerator.

- Here is the syntax that is used to create the data mart definition and create the data mart. The data mart definition is based on the database tables that you create with the ifx_probe2Mart() procedure.

```
ifx_createMart('accelerator_name','data_mart_name','probe_database')
```

accelerator_name

The name of the accelerator where you want to create the data mart.

data_mart_name

The name of the data mart. Valid data mart names have a maximum of 128 characters. Data marts names must be unique in an accelerator.

probe_database

The name of the database that contains the data mart schema tables. If you omit the `probe_database` parameter, the data mart schema tables must be in the current database.

Usage

You must create an accelerator before you can run the ifx_createMart() function.

Return value

The ifx_createMart() function returns the text string "The operation was completed successfully." or an error message.

Example

Examples

This example shows how to create a data mart that is based on a data mart definition.

```
EXECUTE FUNCTION ifx_createMart('MyAccelerator', filetoclob
('orders_customer_mart.xml', 'client'));
```

This example shows how to create the data mart definition and create the data mart that is based on the database tables that you create with the ifx_probe2Mart() procedure.

```
EXECUTE FUNCTION ifx_createMart('MyAccelerator', 'customermart1', 'stores_marts');
```

Related information

[Workload analysis on page 100](#)

ifx_def2Mart() procedure

The ifx_def2Mart() procedure uses the ifx_getMartdef() function to retrieve the data mart definition from an accelerator. It then uses the ifx_xml2Mart() procedure to store the data mart definition XML file in the data mart schema tables.

Syntax

```
ifx_def2Mart('accelerator_name','data_mart_name',
'database_name'
)
```

accelerator_name

The name of the accelerator.

data_mart_name

The name of the data mart.

database_name

If you include the database name, the procedure checks that all the tables and columns that are contained in the data mart definition are present in the database.

Usage

The ifx_def2Mart() procedure retrieves the data mart definition from the accelerator and stores it in the data mart schema tables. The following table lists the data mart schema tables.

Table 13. Data marts schema tables

Table	Description
'informix'.iwa_marts	Names of the data mart definitions
'informix'.iwa_tables	All of the tables that are used in any data mart definition
'informix'.iwa_columns	All of the columns that are used in any data mart definition

Table 13. Data marts schema tables (continued)

Table	Description
'informix'.iwa_mtabs	Tables for a specific data mart definition
'informix'.iwa_mcols	Columns for a specific data mart definition
'informix'.iwa_mrefs	References (join descriptors) of a specific data mart definition
'informix'.iwa_mrefcols	Reference columns (join predicates) of a specific data mart definition

Example**Examples**

This example shows how to retrieve the definition of data mart 'Datamart1' from the accelerator 'MyAccelerator', and then store it in the data mart schema tables of a newly created database 'martSchemaDB'. The procedure checks that the tables and columns that are referenced in the data mart definition exist in database 'database1'.

```
CREATE DATABASE 'martSchemaDB';
EXECUTE procedure ifx_def2Mart('MyAccelerator', 'Datamart1', 'database1');
```

Related reference

[ifx_getMartdef\(\) function on page 145](#)

[ifx_xml2Mart\(\) procedure on page 169](#)

ifx_dropMart() function

The ifx_dropMart() function drops (removes) the data mart from an accelerator.

Syntax

```
ifx_dropMart('accelerator_name','data_mart_name')
```

accelerator_name

The name of the accelerator that contains the data mart.

data_mart_name

The name of the data mart that you want to drop.

Usage

If you are running other operations on the same data mart, such as loading the data mart, the other operation will fail. The ifx_dropMart() function does not wait until the other operations complete.

Return value

The ifx_dropMart() function returns the text string "The operation was completed successfully." or an error message.

Example

Examples

This example shows how to drop a data mart.

```
EXECUTE FUNCTION ifx_dropMart('MyAccelerator', 'Datamart1');
```

Related reference

[ifx_removeDWA\(\) function on page 156](#)

ifx_dropPartMart() function

The `ifx_dropPartMart()` function removes the data in one partition in a fragmented table or all of the data from a non-fragmented table.

Syntax

The data mart must be in an `Active` state.

execute function `ifx_dropPartMart`

```
(  
'accelerator_name'  
,  
'data_mart_name'  
,  
'table_owner'  
,  
'table_name'  
,  
'partition'  
)
```

Parameters

accelerator_name

The name of the accelerator

data_mart_name

The name of the data mart.

table_owner

The owner of the table

table_name

The name of the table

partition

The name of the partition in a fragmented table or the name of the non-fragmented table. The value can also be a partition number that is represented as a string value. You can use the `oncheck -pt` command to look up the partition name, the table name, or the partition number.

Usage

If you want to refresh data in a partition or table, run the `ifx_dropPartMart()` function to remove the stale data, and then run the `loadPartMart()` function to load new data. If you want to remove a partition, run the `ifx_dropPartMart()` function to remove the data, and then detach the partition from the database.

If you plan to use partial data refresh for data marts, including trickle feed, the value of the `MAX_PDQPRIORITY` configuration parameter must be set to 50 or greater during the initial data load and partial data refresh.



Important: Do not run the `ifx_dropPartMart()` function while trickle feed is enabled.

Return values

The `ifx_dropPartMart()` function returns the text string `"The operation was completed successfully."` or an error message.

Example**Examples**

The following example drops the data in partition `'part1'` in the fragmented table `'tab22'`.

```
execute function ifx_dropPartMart('myAccelerator','myMart','user10','tab22',
'part1');
```

The following example drops the data in partition `'0x100124'` in the fragmented table `'tab22'`.

```
execute function ifx_dropPartMart('myAccelerator','myMart','user10','tab22',
'0x100124');
```

The following example drops the data in the non-fragmented table `'tab22'`. In this example, the table name `'tab22'` is used for both the `table_name` parameter and for the `partition` parameter.

```
execute function ifx_dropPartMart('myAccelerator','myMart','user10','tab22',
'tab22');
```

Related information

[Update a data mart on page 114](#)

ifx_genMartDef() function

The `ifx_genMartDef()` function returns a character large object (CLOB) that contains the data mart definition in XML format.

Syntax

ifx_genMartDef('data_mart_name')

data_mart_name

The name of the data mart.

Usage

You can either issue the ifx_genMartDef() function by itself, or incorporate it as a parameter within the LOTOFILE() function. Using the ifx_genMartDef() function with the LOTOFILE() function places the CLOB into an operating system file.

Examples

Use the ifx_genMartDef() function as a parameter within the LOTOFILE() function:

```
EXECUTE FUNCTION LOTOFILE(ifx_genMartDef('salesmart'),
  'salesmart.xml!','client');
```

The following example generates the data mart definition for `salesmart`. The resulting CLOB is used as a parameter within the LOTOFILE() function. The LOTOFILE() function stores the resulting CLOB in an operating system file named `salesmart.xml` on the `client` computer.

```
SELECT lotofile(ifx_genMartDef('salesmart'),'salesmart.xml!','client')
FROM iwa_marts WHERE martname='salesmart';
```

ifx_getdwaMetrics() function

The ifx_getdwaMetrics() function returns metrics about an accelerator.

Syntax

ifx_getdwaMetrics('accelerator_name')

accelerator_name

The name of the accelerator.

Usage

The ifx_getdwaMetrics() function returns a set of unnamed rows. Each row corresponds to a metric of the accelerator. The rows have the following columns:

group

The group that the metric belongs to. The possible values are `'Cluster'`, `'Coordinator'`, or `'Worker'`.

The `Cluster` group has metrics about hardware clusters.

The `Coordinator` group has metrics about coordinator nodes.

The `Worker` group has metrics about worker nodes.

name

The name of the metric. For example, `'DWAWNODE'`.

desc

The description of the metric. For example, 'Number of active worker nodes'.

unit

The unit of the metric. For example, 'nodes', 'queries', 'cores', 'MB', 'ms', '%', Or 'BogoMIPS'.

value

The value of the metric represented as either a text or numeric value.

To view the complete set of rows that are returned by `ifx_getdwaMetrics()` function sorted by group and name, but not including the value, use the following statement:

```
SELECT c.group,c.name,c.desc,c.unit FROM table(ifx_getdwaMetrics('myAccelerator'))
(c) order by 1,2;
```

The following table shows the groups and their associated metric names, descriptions, and units.

Table 14. Groups and their associated metric names, descriptions, and units of the `ifx_getdwaMetrics` function

Group	Metric name	Description	Unit
Cluster	DWASTATE	Cluster state	The unit is one of the following states: <ul style="list-style-type: none"> • Initializing • Fully operational • Recovery mode • Recovering • Error • Maintenance • Maintenance pending • Busy
Cluster	DWACPCB M	Cluster processing capacity	BogoMIPS
Cluster	DWACPCP	Number of cores within the cluster	cores
Cluster	DWASTOR A	Available storage on disk	MB
Cluster	DWASTOR U	Used storage on disk	MB

Table 14. Groups and their associated metric names, descriptions, and units of the ifx_getdwaMetrics function**(continued)**

Group	Metric name	Description	Unit
Coordinator or	DWACNOD E	Number of active coordinator nodes	nodes
Coordinator or	DWACCPU	Average CPU utilization on the coordinator nodes	%
Coordinator or	DWACPMU A	Physical memory available on the coordinator nodes	MB
Coordinator or	DWACPMU U	Physical memory in use on the coordinator nodes	MB
Coordinator or	DWASFREQ	Number of successful query requests since the cluster was started	queries
Coordinator or	DWAFFREQ	Number of failed queries since the cluster was started	queries
Coordinator or	DWAISREQ	Number of failed query requests due to an invalid state	queries
Worker	DWAWNOD E	Number of active worker nodes	nodes
Worker	DWAWCPU	Average CPU utilization on the worker nodes	%
Worker	DWAWPMU A	Physical memory available on the worker nodes	MB
Worker	DWAWPMU U	Physical memory in use on worker nodes	MB
Worker	DWAQQUE L	Average query queue length	queries
Worker	DWAQUEH M	Query queue length high-water mark	queries
Worker	DWAWSMD A	Shared memory available	MB
Worker	DWAWSMD U	Average shared memory used	MB

Table 14. Groups and their associated metric names, descriptions, and units of the ifx_getdwaMetrics function

(continued)

Group	Metric name	Description	Unit
Worker	DWAWSMD M	Maximum shared memory used	MB
Worker	DWADIMRT	The ratio between replicated and distributed data	%
Worker	DWAAQQW T	Average query queue wait time	ms (microseconds)
Worker	DWAQWTH W	Query queue wait time high-water mark	ms (microseconds)

Example**Examples**

The following example shows how to convert the returned result set into a table expression.

```
SELECT c.* FROM TABLE(ifx_getdwaMetrics('MyAccelerator')) (c);
```

The following example shows how to select all metrics for the worker nodes.

```
SELECT c.* FROM TABLE(ifx_getdwaMetrics('MyAccel')) (c)
WHERE c.group='Worker';
```

The following example shows how to select the value and the unit of the metric that is named 'DWAWNODE'.

```
SELECT c.value, c.unit FROM TABLE(ifx_getdwaMetrics('MyAccel')) (c)
WHERE c.name='DWAWNODE';
```

ifx_getMartdef() function

The ifx_getMartdef() function retrieves the data mart definition from an accelerator and returns it as a character large object (CLOB) in XML format.

Syntax

```
ifx_getMartdef('accelerator_name','data_mart_name')
```

accelerator_name

The name of the accelerator that contains the data mart.

data_mart_name

The name of the data mart.

Usage

You can store the data mart definition in a file by using the LOTOFIELD() function, and then edit the XML definition. For example, you can change the data mart name, the tables or the columns, and then create a new data mart.

Return value

The ifx_getMartdef() function returns the value of the data mart definition. For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<dwa:martModel xmlns:dwa="http://www.ibm.com/xmlns/prod/dwa" version="1.0">
  <mart name="orders_customer_mart">
    <table name="customer" schema="informix" isFactTable="false" >
      <column name="customer_num"/>
      <column name="fname"/>
      <column name="lname"/>
      <column name="state"/>
    </table>
    <table name="orders" schema="informix" isFactTable="true" >
      <column name="customer_num"/>
      <column name="ship_date"/>
      <column name="ship_weight"/>
    </table>
    <reference
      referenceType="LEFTOUTER"
      isRuntimeJoin="true"
      parentCardinality="1"
      dependentCardinality="n"
      dependentTableSchema="informix"
      dependentTableName="orders"
      parentTableSchema="informix"
      parentTableName="customer">
      <parentColumn name="customer_num"/>
      <dependentColumn name="customer_num"/>
    </reference>
  </mart>
</dwa:martModel>
```

Example

Examples

This example shows how to retrieve the data mart definition from the accelerator, and then store it in a file on the client computer.

```
EXECUTE FUNCTION lotofield(ifx_getMartdef('MyAccelerator', 'Datamart1'),
  'Datamart1.xml!','client');
```

ifx_getMartStat() function

The ifx_getMartStat() function returns the status of a single data mart as an unnamed row.

Syntax

```
ifx_getMartStat('accelerator_name ','data_mart_name')
```

accelerator_name

The name of the accelerator.

data_mart_name

The name of the data mart.

Usage

The `ifx_getMartStat()` function returns an unnamed row that contains the status for the data mart in the accelerator. The content of the row is the same as the content in the rows for the `ifx_listMarts()` function, which returns the status for all the data marts in an accelerator.

The row has the following columns:

name

The name of the data mart.

status

The status of the data mart. The **status** can have one of the following values:

Active

The data mart is available for query processing. You cannot use the `loadMart()` function to load the data mart when the data mart is in the `Active` state.

LoadInProgress

The data mart is being loading with data. It is not yet available for query processing.

LoadPending

The data mart has been defined successfully, but it is not yet being loading with data. It is not available for query processing.

Disabled

The data mart is deactivated. It is not available for query processing.

memory

The memory that is consumed by the data mart in MB.

lastload

The timestamp of last time that the data mart was loaded successfully.

Example

Examples

The following example shows how to retrieve the unnamed row that contains the status of a single data mart.

```
EXECUTE FUNCTION ifx_getMartStat('MyAccelerator','Data_mart1');
```

The following example shows how to convert the returned result row into a table expression.

```
SELECT c.* FROM TABLE(ifx_getMartStat('MyAccelerator','Data_mart1')) (c);
```

Related reference

[ifx_listMarts\(\) function on page 148](#)

[ifx_setMart\(\) function on page 157](#)

ifx_listMarts() function

The `ifx_listMarts()` function returns a set of unnamed rows that contain the status for all the data marts in an accelerator.

Syntax

```
ifx_listMarts(accelerator_name)
```

accelerator_name

The name of the accelerator.

Usage

The `ifx_listMarts()` function returns a set of unnamed rows that contain the status for each data mart in an accelerator. The content of the rows is the same as the `ifx_getMartStat()` function, which returns the status of a single data mart.

Each row corresponds to one data mart. The rows have the following columns:

name

The name of the data mart.

status

The status of the data mart. The **status** can have one of the following values:

Active

The data mart is available for query processing. You cannot use the `loadMart()` function to load the data mart when the data mart is in the `Active` state.

LoadInProgress

The data mart is being loading with data. It is not yet available for query processing.

LoadPending

The data mart has been defined successfully, but it is not yet being loading with data. It is not available for query processing.

Disabled

The data mart is deactivated. It is not available for query processing.

memory

The memory that is consumed by the data mart in MB.

lastload

The timestamp of last time that the data mart was loaded successfully.

Example**Examples**

The following example shows how to retrieve the set of unnamed rows that contain the status for all the data marts in an accelerator.

```
EXECUTE FUNCTION ifx_listMarts('MyAccelerator');
```

The following example shows how to convert the returned result set into a table expression.

```
SELECT c.* FROM TABLE(ifx_listMarts('MyAccelerator')) (c);
```

The following example shows how to count the number of data marts, and then group them by status.

```
SELECT c.status,count(*) FROM TABLE(ifx_listMarts('MyAccelerator')) (c)
      group by 1;
```

The following example shows how to calculate the amount of memory that has been used by the data marts that have been loaded within the last seven days.

```
SELECT sum(c.memory) FROM TABLE(ifx_listMarts('MyAccelerator')) (c)
      where c.lastload > TODAY - 7;
```

Related reference

[ifx_setMart\(\) function on page 157](#)

[ifx_getMartStat\(\) function on page 146](#)

ifx_loadMart() function

The `ifx_loadMart()` function populates an empty data mart in an accelerator with data. The data is copied from the database tables that are referenced in the data mart definition.

Syntax

```
ifx_loadMart('accelerator_name','data_mart_name','
```

```
TABLE MART NONE
```

```
')
```

accelerator_name

The name of the accelerator that contains the data mart.

data_mart_name

The name of the data mart to load.

The locking mode

This parameter protects the data against changes during the load operation to ensure data consistency. The parameter is case-sensitive. Choose one of these values:

- **TABLE:** Protects only the table that is being loaded against changes. Each table is locked in share mode while it loads with data. The loaded data is consistent within each table, but not necessarily across different tables.
- **MART:** Protects all the tables that are referenced by the data mart against changes during the load operation. All the tables are locked in share mode until the data mart is loaded. The loaded data is consistent from all of the tables. All of the other user sessions are blocked from changing the data in the tables that are loading.
- **NONE:** No locking. The data is read from the different tables without locking them. Other user sessions can change the data during the load operation. As a result, the loaded data might be inconsistent. Data inconsistencies might be acceptable if the purpose of the data mart is to create statistics and discover trends, rather than to find the exact values of specific data rows.

Usage

The data mart must be in the `LoadPending` or the `Disabled` state before you run the `ifx_loadMart()` function. You can use the `ifx_getMartStat()` function to determine the state of the data mart.

After a successful load, the data mart is in the `Active` state and it can be used for query acceleration. The `ifx_loadMart()` function loads the data mart tables sequentially. However, it extracts the data from a single table in parallel, if possible.



Important: To run the `ifx_loadMart()` function, you must be logged in as user **informix** or you must have been granted resource permissions. For example, to grant resource permissions to user Bob:

```
grant resource to Bob;
```

Return value

The `ifx_loadMart()` function returns the text string `"The operation was completed successfully."` or an error message.



Note: Use the SQL statement `set environment use_dwa 'loadpdq <value>';` in the same database session before calling the `ifx_loadMart()` function to set the PDQPRIORITY for the database sessions that extract the data from the database tables. The value for parameter "loadpdq" is a positive integer between 0 and 100 (inclusive), default being 2.

Example**Examples**

This example shows how to load a data mart.


```
EXECUTE FUNCTION ifx_loadMart('MyAccelerator', 'Datamart1', 'MART');
```

Related reference

[ifx_getMartStat\(\) function on page 146](#)

ifx_loadPartMart() function

The `ifx_loadPartMart()` function loads the data into one partition in a fragmented table or into a non-fragmented table.

Syntax

The data mart must be in an `Active` state.

execute function `ifx_loadPartMart`

```
(
'accelerator_name'
,
'data_mart_name'
,
'table_owner'
,
'table_name'
,
'partition'
)
```

Parameters

accelerator_name

The name of the accelerator

data_mart_name

The name of the data mart.

table_owner

The owner of the table

table_name

The name of the table

partition

The name of the partition in a fragmented table or the name of the non-fragmented table. The value can also be a partition number that is represented as a string value. You can use the `oncheck -pt` command to look up the partition name, the table name, or the partition number.

Usage

Run the `ifx_loadPartMart()` function to refresh data in a partition or table after you run the `ifx_dropPartMart()` function. Include the same parameter values in both functions.

If you plan to use partial data refresh for data marts, including trickle feed, the value of the `MAX_PDQPRIORITY` configuration parameter must be set to 50 or greater during the initial data load and partial data refresh.



Important: Do not run the `ifx_loadPartMart()` function while trickle feed is enabled.

Return values

The `ifx_loadPartMart()` function returns the text string `"The operation was completed successfully."` or an error message.



Note: Use the SQL statement `set environment use_dwa 'loadpdq <value>;` in the same database session before calling the `ifx_loadPartMart()` function to set the `PDQPRIORITY` for the database sessions that extract the data from the database tables. The value for parameter "loadpdq" is a positive integer between 0 and 100 (inclusive), default being 2.

Example

Examples

The following example loads the data in partition `'part1'` of the fragmented table `'tab22'`.

```
execute function ifx_loadPartMart('myAccelerator','myMart','user10','tab22',
'part1');
```

The following example loads the data in partition `'0x100124'` of the fragmented table `'tab22'`.

```
execute function ifx_loadPartMart('myAccelerator','myMart','user10','tab22',
'0x100124');
```

The following example loads the data in the non-fragmented table `'tab22'`. In this example, the table name `'tab22'` is used for both the `table_name` parameter and for the `partition` parameter.

```
execute function ifx_loadPartMart('myAccelerator','myMart','user10','tab22',
'tab22');
```

Related information

[Update a data mart on page 114](#)

ifx_probe2Mart() procedure

The `ifx_probe2Mart()` procedure converts the data that is gathered from probing into a data mart definition.

Syntax

```
ifx_probe2Mart('database ','data_mart_name'
,sqlid
)
```

database

The name of the database that contains the data warehouse. This is the warehouse database on which the workload queries are run. See **Usage**.

data_mart_name

The name that you want to use for the data mart definition. The name is also the name of the data mart that is created later, based on the data mart definition. If the name you specify is an existing data mart definition, the probing data is merged into the already existing data mart definition. If the data mart definition you specify does not exist, the data mart definition is created.

sqlid

Optional. The ID of the query SQL statement, which identifies the probing data from that query. If the *sqlid* is not provided, all of the probing data from the specified database is added to the data mart definition.

Usage

The `ifx_probe2Mart()` procedure should be run from a different database than the warehouse database. This separate database must be a logging database. You can use a test database, if the test database is already a logging database, or you can create a different database that keeps these tables separated from your other tables.

When the procedure is run, the probing data is processed and stored in the logging database in a set of permanent tables.

The tables keep the data mart definition in a relational format. The tables are automatically created when the probing data is processed into a data mart definition for the first time.

The `ifx_probe2Mart()` procedure creates a data mart definition by converting the probing data and inserting rows into the following data mart schema tables.

Table 15. Data marts schema tables

Table	Description
'informix'.iwa_marts	Names of the data mart definitions
'informix'.iwa_tables	All of the tables that are used in any data mart definition
'informix'.iwa_columns	All of the columns that are used in any data mart definition
'informix'.iwa_mtabs	Tables for a specific data mart definition
'informix'.iwa_mcols	Columns for a specific data mart definition
'informix'.iwa_mrefs	References (join descriptors) of a specific data mart definition
'informix'.iwa_mrefcols	Reference columns (join predicates) of a specific data mart definition

Examples

To convert, or merge, all of the probing data into a data mart definition, use this form of the syntax:

```
EXECUTE PROCEDURE ifx_probe2Mart('database', 'mart_name');
```

For example, to generate a data mart definition named `salesmart` from all of probing data that is available for the database `sales`, use this statement:

```
EXECUTE PROCEDURE ifx_probe2Mart('sales', 'salesmart');
```

You can also merge the probing data from a specific query into a data mart definition. You need to look up the SQL ID number of the query that was captured by SQL tracing. SQL tracing must be ON to designate data from specific queries. Queries are identified by a statement ID.

For example, to merge the probing data from SQL statement 8372 into the data mart definition `salesmart`, run this command:

```
EXECUTE PROCEDURE ifx_probe2Mart('sales', 'salesmart', 8372);
```

To create a data mart definition from queries that run longer than 10 seconds, use this SQL statement:

```
SELECT ifx_probe2Mart('sales','salesmart',sql_id)
FROM sysmaster:syssqltrace
WHERE sql_runtime > 10;
```

ifx_refreshMart() function

The `ifx_refreshMart()` function checks for data changes in individual partitions and refreshes the changed partitions.

Syntax

```
ifx_refreshMart('accelerator_name','data_mart_name'
,
NULL locking_mode
,
,
NULL threshold
,
)
```

accelerator_name

The name of the accelerator that contains the data mart.

data_mart_name

The name of the data mart to refresh.

locking_mode

Specifies the level of data protection during the refresh. Use one of the following values, which must be in uppercase:

- NULL = The locking level that you specified the last time you ran the `ifx_loadMart()` function is used
- TABLE = Protects only the table that is being refreshed against changes. Each table is locked in share mode while it refreshes with data. The refreshed data is consistent within each table, but not necessarily across different tables.
- MART = Protects all the tables that are referenced by the data mart against changes during the refresh operation. All the tables are locked in share mode until the data mart is refreshed. The refreshed data is consistent from all of the tables. All of the other user sessions are blocked from changing the data in the tables that are refreshing.
- NONE = No locking. The data is read from the different tables without locking them. Other user sessions can change the data during the refresh operation. As a result, the refreshed data might be inconsistent. Data inconsistencies might be acceptable if the purpose of the data mart is to create statistics and discover trends, rather than to find the exact values of specific data rows.

threshold

Reserved for future use. Set to `'NULL'`.

Usage

After you load a data mart, you can run the `ifx_refreshMart()` function to refresh the data. For a high-availability cluster, you can run the `ifx_refreshMart()` function on the primary server or on the secondary server.

If a fragment is detached from the dimension or fact table, the `ifx_refreshMart()` function drops the corresponding partition from the data mart. If a fragment is attached to the dimension or fact table, the `ifx_refreshMart()` function adds the partition to the data mart.

If trickle feed is enabled, the trickle feed task refreshes the fact tables and runs the `ifx_refreshMart()` function to refresh the dimension tables.

The `ifx_refreshMart()` function does not update the data mart definition. If a table is added or dropped from the database server, you must drop the data mart, update the data mart definition, and then re-create the data mart.

If you plan to use partial data refresh for data marts, including trickle feed, the value of the `MAX_PDQPRIORITY` configuration parameter must be set to 50 or greater during the initial data load and partial data refresh.

Return value

The `ifx_refreshMart()` function returns the text string `"The operation was completed successfully."` or an error message.



Note: Use the SQL statement `set environment use_dwa 'loadpdq <value>';` in the same database session before calling the `ifx_refreshMart()` function to set the `PDQPRIORITY` for the database sessions that extract the data from the database tables. The value for parameter "loadpdq" is a positive integer between 0 and 100 (inclusive), default being 2.

Example

Examples

The following statement refreshes a data mart named **Datamart1** in an accelerator named **MyAccelerator** and specifies a locking mode of MART:

```
EXECUTE FUNCTION ifx_refreshMart('MyAccelerator', 'Datamart1', 'MART', 'NULL');
```

Related information

[Update a data mart on page 114](#)

ifx_removeDWA() function

The `ifx_removeDWA()` function removes (drops) an accelerator from the accelerator server. This function removes the connectivity information from the `sqlhosts` file in the Informix® database server. The data marts that are associated with the accelerator are not removed, but they are not accessible. Drop the data marts before you drop the accelerator.

Syntax

```
ifx_removeDWA(accelerator_name)
```

accelerator_name

The name of the accelerator.

Usage



Important: To run the `ifx_removeDWA()` function, you must be logged in as user `informix` and you must have write access to the `sqlhosts` file and to the directory where the `sqlhosts` file is located.

You do not need a connection to the accelerator server to run the `ifx_removeDWA()` function.

Return value

The `ifx_removeDWA()` function returns the text string `"The operation was completed successfully."` or an error message.

Example

Examples

This example shows how to drop an accelerator.

```
EXECUTE FUNCTION ifx_removeDWA('MyAccelerator');
```

Related reference

[ifx_dropMart\(\) function on page 139](#)

ifx_removeTrickleFeed() function

The ifx_removeTrickleFeed() function stops trickle feed so that data in the data mart is no longer continuously refreshed.

Syntax

```
ifx_removeTrickleFeed(' accelerator_name','data_mart_name '
)
```

accelerator_name

The name of the accelerator that contains the data mart.

data_mart_name

The name of the data mart.

Usage

If trickle feed is enabled, the ifx_removeTrickleFeed() function disables it. The ifx_removeTrickleFeed() function removes the ifx_TrickleFeed_load_ID task for the specified data mart in the Scheduler.

Return value

The ifx_removeTrickleFeed() function returns the text string "The operation was completed successfully." or an error message.

Example

Examples

The following statement stops the automatic refreshing of data for a data mart named **Datamart1** in an accelerator named **MyAccelerator**:

```
EXECUTE FUNCTION ifx_removeTrickleFeed('MyAccelerator', 'Datamart1');
```

ifx_setMart() function

The ifx_setMart() function changes the state of a data mart from **Active** to **Disabled** or from **Disabled** to **Active**.

Syntax

```
ifx_setMart('accelerator_name','data_mart_name',
```

```
,
```

OFF ON

```
,
```

```
)
```

accelerator_name

The name of the accelerator.

data_mart_name

The name of the data mart.

OFF

Changes the state of the data mart state from `Active` to `Disabled`. This parameter is case-sensitive.

ON

Changes the state of the data mart state from `Disabled` to `Active`. This parameter is case-sensitive.

Usage

To determine the state of a data mart, use the `ifx_getMartStat()` function. The `ifx_setMart()` function returns an error if the data mart is in a state other than `Active` or `Disabled`.

When the data mart is in the `Active` state, the data mart is available for query processing.

When the data mart is in the `Disabled` state, the data mart is not available for query processing.

Return value

The `ifx_setMart()` function returns the text string "The operation was completed successfully." or an error message.

Example**Examples**

The following example shows how to change a data mart that is in the `Active` state to the `Quiesced` state.

```
EXECUTE FUNCTION ifx_setMart('MyAccelerator', 'Datamart1' 'OFF');
```

The following example shows how to change a data mart that is in the `Disabled` state to the `Active` state.

```
EXECUTE FUNCTION ifx_setMart('MyAccelerator', 'Datamart1' 'ON');
```

Related reference

[ifx_listMarts\(\) function on page 148](#)

[ifx_getMartStat\(\) function on page 146](#)

ifx_setupDWA() function

The `ifx_setupDWA()` function creates an accelerator in an accelerator server. This function creates the connection between the database server and the accelerator server. You can also use this function to renew the connectivity information for an accelerator.

Syntax

```
ifx_setupDWA('accelerator_name','ip','port_number','pairing_code')
```


accelerator_name

The name of the accelerator. You must use a name that does not exist in the accelerator server. The accelerator name has a maximum length of 128 characters.

ip

The IP address of the accelerator server. You obtain this value by running the `ondwa getpin` command. If you are renewing the connection for an accelerator, this value must be `NULL`.

port_number

The service port number of the accelerator server. You obtain this value by running the `ondwa getpin` command. If you are renewing the connection for an accelerator, this value must be `NULL`.

pairing_code

The pairing code. You obtain this value by running the `ondwa getpin` command. If you are renewing the connection for an accelerator, this value must be `NULL`.

Usage

Important: To run the `ifx_setupDWA()` function, you must be logged in as user `informix` and you must have write access to the `sqlhosts` file and to the directory where the `sqlhosts` file is located.

Return value

The `ifx_setupDWA()` function returns the text string `"The operation was completed successfully."` or an error message.

Example**Examples**

The following example shows how to create an accelerator with the name `'MyAccelerator'`. The IP address, port number, and pairing code are the results of running the `ondwa getpin` command.

```
EXECUTE FUNCTION ifx_setupDWA('MyAccelerator', '127.0.0.1', '21022', '1234');
```

The following example shows how to renew the connection for an accelerator with the name `'MyAccelerator'`. The last three parameters must be specified as `NULL`.

```
EXECUTE FUNCTION ifx_setupDWA('MyAccelerator', NULL, NULL, NULL);
```

Related reference

[ondwa getpin command on page 91](#)

ifx_setupTrickleFeed() function

The `ifx_setupTrickleFeed()` function enables the continuous refreshing of data in the fact and dimension tables of the data mart as the data changes in the database.

Syntax

```
ifx_setupTrickleFeed('accelerator_name ','data_mart_name'
,buffertime
)
```

accelerator_name

The name of the accelerator that contains the data mart.

data_mart_name

The name of the data mart.

buffertime

An integer that represents the time interval between refreshes and whether dimension tables are refreshed. For best results, set to at least 10 seconds.

-1 through -8639999 = The absolute value is used as the number of seconds between fact table refreshes. Dimension tables are not refreshed.

1 through 8639999 = The number of seconds between refreshes of the fact and dimension tables.

Usage

After you load a data mart, run the `ifx_setupTrickleFeed()` function to enable trickle feed. For a high-availability cluster, run the `ifx_setupTrickleFeed()` function on the primary server.

If you plan to use partial data refresh for data marts, including trickle feed, the value of the `MAX_PDQPRIORITY` configuration parameter must be set to 50 or greater during the initial data load and partial data refresh.

As data is inserted into the database, the data in the specified data mart is refreshed at the frequency specified by the `buffertime` parameter. Between refreshes, the new data that is inserted into the fact tables is stored in the `$INFORMIXDIR/tmp/trickleFeed` directory on the database server. A small value of the `buffertime` parameter, such as 60, has the following advantages:

- The data in the data mart is current.
- Loading the data has a minimal impact on accelerated queries.
- A small amount of disk space is needed to store the updates on the database server between refreshes.

Choose the value of the `buffertime` parameter carefully to suit your business needs.

Set the `buffertime` parameter to a negative number when you want to refresh insertions into the fact tables but ignore changes to the dimension tables.

The `ifx_setupTrickleFeed()` function adds the `ifx_TrickleFeed_load_ID` task to the Scheduler, where `ID` is a unique number. Every data mart for which you enable trickle feed has a trickle feed task in the Scheduler. You can determine if trickle feed is enabled for a data mart by querying the `ph_task` table in the `sysadmin` database. The `tk_name` column contains the name of the task and the `tk_description` column contains the description, which lists the data mart and accelerator names. If a task description includes the relevant data mart name, trickle feed is running for that data mart.



Important: You must periodically disable trickle feed and reload the data mart. The `ifx_TrickleFeed_load_ID` task in the Scheduler stops running after 32000 refreshes.

Trickle feed captures the data that is inserted into the fact tables and all types of data changes to the dimension tables. Use other methods to make any other changes to the data or the data mart. The `ifx_setupTrickleFeed()` function does not make the following changes:

- Update or delete data in the fact tables. Typically, changes to existing data is not allowed in warehouses. If data is updated, disable trickle feed and run the `ifx_refreshMart()` function.
- Attach or detach a partition in the data mart when a fragment is attached to or detached from a fact or dimension table. You must disable trickle feed and run the `ifx_refreshMart()` function or reload the data mart.
- Update the data mart definition. If a table is added or dropped from the database server, you must disable trickle feed, drop the data mart, update the data mart definition, and then re-create the data mart.

Do not do any of the following tasks while trickle feed is enabled:

- Refresh a partition by running the `ifx_loadPartMart()` function.
- Drop a partition by running the `ifx_dropPartMart()` function.
- Load the data mart by running the `ifx_loadMart()` function.
- Drop the data mart by running the `ifx_dropMart()` function. The `ifx_dropMart()` function disables trickle feed.

To disable trickle feed, run the `ifx_removeTrickleFeed()` function.

Return value

The `ifx_setupTrickleFeed()` function returns the text string "The operation was completed successfully." or an error message.

Example

Example 1: Start trickle feed

The following statement starts refreshing data every 60 seconds for a data mart named **Datamart1** in an accelerator named **MyAccelerator**:

```
EXECUTE FUNCTION ifx_setupTrickleFeed('MyAccelerator', 'Datamart1', 60);
```

Example

Example 2: Determine if trickle feed is running

The following SQL statement returns every trickle feed task name and description:

```
SELECT * FROM sysadmin::ph_task WHERE tk_name MATCHES 'ifx_TrickleFeed_load*';
```

The results of this query show that trickle feed is running for the data mart named **Datamart1** in an accelerator named **MyAccelerator**:

```
...
tk_id          84
tk_name        ifx_TrickleFeed_load_16
```

```
tk_description      trickle feed loader for data mart Datamart1@MyAccelerator
...
```

Related information

[Update a data mart on page 114](#)

ifx_TSDW_createWindow() function

The ifx_TSDW_createWindow function creates a time window to limit the amount of time series data in a time series virtual table of a data mart.

Syntax

```
ifx_TSDW_createWindow(accelerator_name varchar(128),
                      datamart_name  varchar(128),
                      table_owner    varchar(32),
                      table_name     varchar(128),
                      begin_index     integer,
                      end_index      integer)

returns lvarchar;

ifx_TSDW_createWindow(accelerator_name varchar(128),
                      datamart_name  varchar(128),
                      table_owner    varchar(32),
                      table_name     varchar(128),
                      begin_stamp     datetime year to fraction(5),
                      end_stamp      datetime year to fraction(5))

returns lvarchar;
```

accelerator_name

The name of the accelerator.

datamart_name

The name of the data mart.

schema_owner

The owner of the table.

table_name

The name of the table.

begin_index

The calendar index that identifies the first virtual partition that is included in the time window. The value must be greater than or equal 0.

end_index

The calendar index that identifies the first virtual partition that is excluded in the time window. The value must be greater than *begin_index*.

begin_stamp

The time stamp that identifies the first virtual partition included in the time window. The value must be greater than or equal to the start time stamp of the calendar that is assigned to the table.

end_stamp

The time stamp that identifies the first virtual partition that is excluded in the time window. The value must identify a virtual partition that is greater than the virtual partition identified by *begin_stamp*.

Usage

Run the `ifx_TSDW_createWindow()` function to create a time window for a time series virtual table in a data mart.

You must create the first time window before the data mart is initially loaded, with the state of the data mart in LoadPending. You can create subsequent time windows either before or after the data mart is initially loaded, with the state of the data mart in LoadPending or Active.

If a time window is created in an active data mart, the time series data is loaded immediately. Depending on the volume of data, this load can take a significant amount of time.

Return values

The `ifx_TSDW_createWindow()` function returns the text string `"The operation was completed successfully."` or an error message.

Example

The following examples create time windows for the time series virtual table named `'informix'. 'ts_data_v'` in data mart named `'datamart_name'` on accelerator named `'demo_dwa'`. A calendar with a start time stamp of `'2010-01-01 00:00:00.000000'` and a pattern of one month is set for this table.

The following example creates a time window that includes January and February 2011:

```
execute function ifx_TSDW_createWindow(
  'demo_dwa', 'datamart_name', 'informix', 'ts_data_v',
  '2011-01'::datetime year to month, '2011-03'::datetime year to month);
```

The following example creates a time window that includes March and April 2011:

```
execute function ifx_TSDW_createWindow(
  'demo_dwa', 'datamart_name', 'informix', 'ts_data_v', 14, 16);
```

Related reference

[ifx_TSDW_dropWindow\(\) function on page 163](#)

Related information

[Limit the size of data with time windows on page 120](#)

ifx_TSDW_dropWindow() function

The `ifx_TSDW_dropWindow` function removes time windows from a time series virtual table in a data mart.

Syntax

```

ifx_TSDW_dropWindow(accelerator_name varchar(128),
                    data_mart_name  varchar(128),
                    table_owner     varchar(32),
                    table_name      varchar(128),
                    begin_index     integer)

returns lvarchar;

ifx_TSDW_dropWindow(accelerator_name varchar(128),
                    data_mart_name  varchar(128),
                    table_owner     varchar(32),
                    table_name      varchar(128),
                    begin_stamp     datetime year to fraction(5))

returns lvarchar;

```

accelerator_name

The name of the accelerator.

datamart_name

The name of the data mart.

table_owner

The owner of the table.

table_name

The name of the table.

begin_index

The calendar index that identifies the first virtual partition that is included in the time window. The value must be greater than or equal 0.

begin_stamp

The time stamp that identifies the first virtual partition included in the time window. The value must be greater than or equal to the start time stamp of the calendar that is assigned to the table.

Usage

Run the ifx_TSDW_dropWindow function to remove a time window for a time series virtual table in a data mart.

You can remove time windows before or after the data mart is initially loaded, when the state of the data mart is LoadPending or Active.

If a time window is removed in an active data mart, the time series data is deleted immediately. Depending on the volume of data, this load can take a significant amount of time.

Return value

The ifx_TSDW_moveWindows() function returns the text string "The operation was completed successfully." or an error message.

Examples

The following examples remove time windows for the time series virtual table `'informix'. 'ts_data_v'` in data mart `'datamart_name'` on accelerator `'demo_dwa'`. A calendar with a start time stamp of `'2010-01-01 00:00:00.00000'` and a pattern of one month is set for this table. The time windows were created previously by using the `ifx_TSDW_createWindow` function.

The following example removes a time window that begins January 2011:

```
execute function ifx_TSDW_dropWindow(
  'demo_dwa', 'datamart_name', 'informix', 'ts_data_v',
  '2011-01'::datetime year to month);
```

The following example removes a time that begins March 2011:

```
execute function ifx_TSDW_createWindow(
  'demo_dwa', 'datamart_name', 'informix', 'ts_data_v', 14);
```

Related reference

[ifx_TSDW_createWindow\(\) function on page 162](#)

Related information

[Limit the size of data with time windows on page 120](#)

ifx_TSDW_moveWindows() function

The `ifx_TSDW_moveWindows` function moves time windows in a time series virtual table within a data mart.

Syntax

```
ifx_TSDW_moveWindows(accelerator_name varchar(128),
                    data_mart_name  varchar(128),
                    table_owner     varchar(32),
                    table_name      varchar(128),
                    num_partitions   integer DEFAULT 1)
returns lvarchar;
```

accelerator_name

The name of the accelerator.

datamart_name

The name of the data mart.

table_owner

The owner of the table.

table_name

The name of the table.

num_partitions

The number of virtual partitions by which the time window is moved. This value can be a positive or negative integer. The default value is 1.

Usage

Run the `ifx_TSDW_moveWindows()` function to move time windows of a time series virtual table within a data mart by a number of virtual partitions.

The time series data for the virtual partitions that are no longer used is deleted from the accelerator. The time series data for the virtual partitions that are within the time windows is loaded to the accelerator.

A positive value for *num_partitions* moves the time windows forward, a negative value moves the time windows backward. When a time window is moved so that a portion of the virtual partitions is no longer within a time stamp of the assigned calendar, the portion outside of the time stamp is removed. When a time window is moved so that none of the virtual partitions are located within a time stamp of the assigned calendar, the entire time window is removed.

If time windows are moved in an active data mart, the time series data is deleted immediately. Depending on the volume of data, this load can take a significant amount of time.

Return values

The `ifx_TSDW_moveWindows()` function returns the text string "The operation was completed successfully." or an error message.

Examples

In this example, the time windows for the time series virtual table `'informix'. 'ts_data_v'` in data mart `'datamart_name'` on accelerator `'demo_dwa'` are moved. A calendar with a start time stamp of `'2010-01-01 00:00:00.00000'` and a pattern of one month is set for this table. A time window is created that contains time series data for January and February 2011.

- To move the time window that contains time series data for January and February 2011 forward by one month:

```
execute function ifx_TSDW_moveWindows(
  'demo_dwa', 'datamart_name', 'informix', 'ts_data_v');
```

The data for January 2011 is deleted on the accelerator, and the data for March 2011 is loaded. The data for February 2011 is unchanged.

- In the previous example, the original time window was moved forward by one month, so that it contains data for February and March 2011. To move this time window forward by three more months:

```
execute function ifx_TSDW_moveWindows(
  'demo_dwa', 'datamart_name', 'informix', 'ts_data_v', 3);
```

The data for February and March 2011 is deleted on the accelerator, and the data for May and June 2011 is loaded.

Related information

[Limit the size of data with time windows on page 120](#)

ifx_TSDW_updatePartition() function

The ifx_TSDW_updatePartition function refreshes the time series data within a time series virtual table in a data mart.

Syntax

```
ifx_TSDW_updatePartition(accelerator_name varchar(128),
                        data_mart_name  varchar(128),
                        table_owner     varchar(32),
                        table_name      varchar(128),
                        calendar_index   integer)

returns lvarchar;

ifx_TSDW_updatePartition(accelerator_name varchar(128),
                        data_mart_name  varchar(128),
                        table_owner     varchar(32),
                        table_name      varchar(128),
                        time_stamp      datetime year to fraction(5))

returns lvarchar;
```

accelerator_name

The name of the accelerator.

datamart_name

The name of the data mart.

table_owner

The owner of the table.

table_name

The name of the table.

calendar_index

The calendar index that identifies the virtual partition that is refreshed. The value must be greater than or equal to 0.

time_stamp

The time stamp that identifies the virtual partition that is refreshed. The value must be greater than or equal to the start time stamp of the calendar that is assigned to the table.

Usage

Run the ifx_TSDW_updatePartition() function to refresh the time series data of a virtual partition within a time series virtual table in a data mart.

If the time series virtual table has time windows, the virtual partition must exist in one of the time windows. If a virtual partition is refreshed in an active data mart, the time series data is refreshed immediately. Depending on the volume of data, this load can take a significant amount of time.

Return values

The `ifx_TSDW_updatePartition()` function returns the text string "The operation was completed successfully." or an error message.

Example

The following example refreshes time series data for the time series virtual table `'informix'. 'ts_data_v'` in data mart `'datamart_name'` on accelerator `'demo_dwa'`. A calendar with a start time stamp of `'2010-01-01 00:00:00.00000'` and a pattern of one month is set for this table.

The following example refreshes the data for January 2011.

```
execute function ifx_TSDW_updatePartition(
  'demo_dwa', 'datamart_name', 'informix', 'ts_data_v',
  '2011-01'::datetime year to month);
```

The following example refreshes the data for March 2011.

```
execute function ifx_TSDW_updatePartition(
  'demo_dwa', 'datamart_name', 'informix', 'ts_data_v', 14);
```

Related information

[Limit the size of data with time windows on page 120](#)

ifx_TSDW_setCalendar() function

The `ifx_TSDW_setCalendar` function assigns a time series calendar to a time series virtual table in a data mart.

Syntax

```
ifx_TSDW_setCalendar(accelerator_name varchar(128),
                    data_mart_name  varchar(128),
                    table_owner     varchar(32),
                    table_name      varchar(128),
                    calendar_name   varchar(255))
returns lvarchar;
```

accelerator_name

The name of the accelerator.

datamart_name

The name of the data mart.

table_owner

The owner of the table.

table_name

The name of the table.

calendar_name

The name of the time series calendar.

Usage

Run the `ifx_TSDW_setCalendar()` function to assign a time series calendar to a time series virtual table in a data mart.

When you assign a calendar, the time series data is grouped into virtual partitions when it is loaded to the accelerator. The data mart must not be initially loaded, in the LoadPending state.

Return value

The `ifx_TSDW_setCalendar()` function returns the text string `"The operation was completed successfully."` or an error message.

Example

The following example assigns a time series calendar `'2010monthly'` to the time series virtual table `'informix'. 'ts_data_v'` in data mart `'datamart_name'` on accelerator `'demo_dwa'`.

```
execute function ifx_TSDW_setCalendar(
  'demo_dwa', 'datamart_name', 'informix', 'ts_data_v', '2010monthly');
```

Related information

[Define virtual partitions by assigning a calendar on page 119](#)

ifx_xml2Mart() procedure

The `ifx_xml2Mart()` procedure converts a data mart definition from XML format into data mart schema tables.

Syntax

```
ifx_xml2Mart(data_mart_definition ,
'database_name'
)
```

data_mart_definition

The data mart definition in XML format (data type CLOB).

database_name

If you include the database name, the procedure checks that all the tables and columns that are contained in the data mart definition XML file are present in the database.

Usage

The `ifx_xml2Mart()` procedure converts a data mart definition that is in XML format into the data mart schema tables. You must run the `ifx_xml2Mart()` procedure in a logged database. If the data mart schema tables do not exist, they are created automatically. The following table lists the data mart schema tables.

Table 16. Data marts schema tables

Table	Description
'informix'.iwa_marts	Names of the data mart definitions
'informix'.iwa_tables	All of the tables that are used in any data mart definition
'informix'.iwa_columns	All of the columns that are used in any data mart definition
'informix'.iwa_mtabs	Tables for a specific data mart definition
'informix'.iwa_mcols	Columns for a specific data mart definition
'informix'.iwa_mrefs	References (join descriptors) of a specific data mart definition
'informix'.iwa_mrefcols	Reference columns (join predicates) of a specific data mart definition

Example**Examples**

This example shows how to convert a data mart definition that is stored in a file into data mart schema tables. The procedure checks that the tables and columns that are referenced in the data mart definition exist in database 'database1'.

```
EXECUTE procedure ifx_xml2Mart(filetoclob('mart.xml', 'client'), 'database1');
```

Troubleshooting**Missing sbpace**

If you do not have a default sbpace created in the Informix® database server and you attempt to install , an error is returned.

You must create and configure a default sbpace in the Informix® database server, set the name of the default sbpace in the SBSPACENAME configuration parameter, and restart the Informix® database server.

Related information

[Preparing the Informix database server on page 72](#)

Ensuring a result set includes the most current data

Occasionally you want a specific query to access the most current data, which is stored on the database server. You can turn off acceleration temporarily to run the query.

About this task

The data that is stored on the accelerator server is a snap-shot of the data on the database server. If you need to ensure that a query result set includes the most current data, turn off acceleration, run the query, and turn on acceleration again:

1. Specify the `SET ENVIRONMENT use_dwa 'accelerate off'` statement at the beginning of the query to turn off acceleration. Setting this variable ensures that the query is processed by the Informix® database server and not processed by the accelerator server.
2. Add the statement `SET ENVIRONMENT use_dwa 'accelerate on'` at the end of the query to activate acceleration again.
3. Run the query.

Sample warehouse schema

The examples use this sample warehouse schema.

SQL statements

The following SQL statements create the tables, indexes, and key constraints for the sample warehouse schema.

```
CREATE TABLE DAILY_FORECAST (
  PERKEY  INTEGER NOT NULL ,
  STOREKEY  INTEGER NOT NULL ,
  PRODKEY  INTEGER NOT NULL ,
  QUANTITY_FORECAST  INTEGER ,
  EXTENDED_PRICE_FORECAST  DECIMAL(16,2) ,
  EXTENDED_COST_FORECAST  DECIMAL(16,2) );

CREATE TABLE DAILY_SALES (
  PERKEY  INTEGER NOT NULL ,
  STOREKEY  INTEGER NOT NULL ,
  CUSTKEY  INTEGER NOT NULL ,
  PRODKEY  INTEGER NOT NULL ,
  PROMOKEY  INTEGER NOT NULL ,
  QUANTITY_SOLD  INTEGER ,
  EXTENDED_PRICE  DECIMAL(16,2) ,
  EXTENDED_COST  DECIMAL(16,2) ,
  SHELF_LOCATION  INTEGER ,
  SHELF_NUMBER  INTEGER ,
  START_SHELF_DATE  INTEGER ,
  SHELF_HEIGHT  INTEGER ,
  SHELF_WIDTH  INTEGER ,
  SHELF_DEPTH  INTEGER ,
  SHELF_COST  DECIMAL(16,2) ,
  SHELF_COST_PCT_OF_SALE  DECIMAL(16,2) ,
  BIN_NUMBER  INTEGER ,
  PRODUCT_PER_BIN  INTEGER ,
  START_BIN_DATE  INTEGER ,
  BIN_HEIGHT  INTEGER ,
  BIN_WIDTH  INTEGER ,
  BIN_DEPTH  INTEGER ,
  BIN_COST  DECIMAL(16,2) ,
  BIN_COST_PCT_OF_SALE  DECIMAL(16,2) ,
```

```

TRANS_NUMBER INTEGER ,
HANDLING_CHARGE INTEGER ,
UPC INTEGER ,
SHIPPING INTEGER ,
TAX INTEGER ,
PERCENT_DISCOUNT INTEGER ,
TOTAL_DISPLAY_COST DECIMAL(16,2) ,
TOTAL_DISCOUNT DECIMAL(16,2) ) ;

CREATE TABLE CUSTOMER (
  CUSTKEY INTEGER NOT NULL ,
  NAME CHAR(30) ,
  ADDRESS CHAR(40) ,
  C_CITY CHAR(20) ,
  C_STATE CHAR(5) ,
  ZIP CHAR(5) ,
  PHONE CHAR(10) ,
  AGE_LEVEL SMALLINT ,
  AGE_LEVEL_DESC CHAR(20) ,
  INCOME_LEVEL SMALLINT ,
  INCOME_LEVEL_DESC CHAR(20) ,
  MARITAL_STATUS CHAR(1) ,
  GENDER CHAR(1) ,
  DISCOUNT DECIMAL(16,2) ) ;

ALTER TABLE CUSTOMER
  ADD CONSTRAINT PRIMARY KEY
  ( CUSTKEY );

CREATE TABLE PERIOD (
  PERKEY INTEGER NOT NULL ,
  CALENDAR_DATE DATE ,
  DAY_OF_WEEK SMALLINT ,
  WEEK SMALLINT ,
  PERIOD SMALLINT ,
  YEAR SMALLINT ,
  HOLIDAY_FLAG CHAR(1) ,
  WEEK_ENDING_DATE DATE ,
  MONTH CHAR(3) ) ;

ALTER TABLE PERIOD
  ADD CONSTRAINT PRIMARY KEY
  ( PERKEY );

CREATE UNIQUE INDEX PERX1 ON PERIOD
  ( CALENDAR_DATE ASC,
  PERKEY ASC );

CREATE UNIQUE INDEX PERX2 ON PERIOD
  ( WEEK_ENDING_DATE ASC,
  PERKEY ASC );

CREATE TABLE PRODUCT (
  PRODKEY INTEGER NOT NULL ,
  UPC_NUMBER CHAR(11) NOT NULL ,
  PACKAGE_TYPE CHAR(20) ,
  FLAVOR CHAR(20) ,
  FORM CHAR(20) ,

```

```

CATEGORY INTEGER ,
SUB_CATEGORY INTEGER ,
CASE_PACK INTEGER ,
PACKAGE_SIZE CHAR(6) ,
ITEM_DESC CHAR(30) ,
P_PRICE DECIMAL(16,2) ,
CATEGORY_DESC CHAR(30) ,
P_COST DECIMAL(16,2) ,
SUB_CATEGORY_DESC CHAR(70) ) ;

ALTER TABLE PRODUCT
ADD CONSTRAINT PRIMARY KEY
( PRODKEY );

CREATE UNIQUE INDEX PRODX2 ON PRODUCT
( CATEGORY ASC,
  PRODKEY ASC );

CREATE UNIQUE INDEX PRODX3 ON PRODUCT
( CATEGORY_DESC ASC,
  PRODKEY ASC );

CREATE TABLE PROMOTION (
  PROMOKEY INTEGER NOT NULL ,
  PROMOTYPE INTEGER ,
  PROMODESC CHAR(30) ,
  PROMOVALUE DECIMAL(16,2) ,
  PROMOVALUE2 DECIMAL(16,2) ,
  PROMO_COST DECIMAL(16,2) ) ;

ALTER TABLE PROMOTION
ADD CONSTRAINT PRIMARY KEY
( PROMOKEY );

CREATE UNIQUE INDEX PROMOX1 ON PROMOTION
( PROMODESC ASC,
  PROMOKEY ASC);

CREATE TABLE STORE (
  STOREKEY INTEGER NOT NULL ,
  STORE_NUMBER CHAR(2) ,
  CITY CHAR(20) ,
  STATE CHAR(5) ,
  DISTRICT CHAR(14) ,
  REGION CHAR(10) ) ;

ALTER TABLE STORE
ADD CONSTRAINT PRIMARY KEY
( STOREKEY );

CREATE INDEX DFX1 ON DAILY_FORECAST ( PERKEY ASC);

CREATE INDEX DFX2 ON DAILY_FORECAST ( STOREKEY ASC);

```

```

CREATE INDEX DFX3 ON DAILY_FORECAST ( PRODKEY ASC);

CREATE INDEX DSX1 ON DAILY_SALES ( PERKEY ASC);

CREATE INDEX DSX2 ON DAILY_SALES ( STOREKEY ASC);

CREATE INDEX DSX3 ON DAILY_SALES ( CUSTKEY ASC);

CREATE INDEX DSX4 ON DAILY_SALES ( PRODKEY ASC);

CREATE INDEX DSX5 ON DAILY_SALES ( PROMOKEY ASC);

ALTER TABLE daily_sales ADD CONSTRAINT FOREIGN KEY (perkey)
  references period(perkey);
ALTER TABLE daily_sales ADD CONSTRAINT FOREIGN KEY (prodkey)
  references product(prodkey);
ALTER TABLE daily_sales ADD CONSTRAINT FOREIGN KEY
  (storekey) references store(storekey);
ALTER TABLE daily_sales ADD CONSTRAINT FOREIGN KEY (custkey)
  references customer(custkey);
ALTER TABLE daily_sales ADD CONSTRAINT FOREIGN KEY (promokey)
  references promotion(promokey);

ALTER TABLE daily_forecast ADD CONSTRAINT FOREIGN KEY (perkey)
  references period(perkey);
ALTER TABLE daily_forecast ADD CONSTRAINT FOREIGN KEY (prodkey)
  references product(prodkey);
ALTER TABLE daily_forecast ADD CONSTRAINT FOREIGN KEY (storekey)
  references store(storekey);

update statistics medium;

```

Sysmaster interface (SMI) pseudo tables for query probing data

The SMI tables provide a way to access probing data in a relational form, which is most convenient for further processing.

The sysmaster database provides the following pseudo tables for accessing probing data:

- For tables: sysprobetables
- For columns: sysprobecolumns
- For join descriptors: sysprobejds
- For join predicates: sysprobejps

The sysprobetables table

The schema for the sysprobetables table is:

```
CREATE TABLE informix.sysprobetables
(
  dbname char(128),  { database name }
  sql_id int8,      { statement id in syssqltrace }
  tabid integer,    { table id }
  fact char(1)      { table is fact table (y/n) }
);
REVOKE ALL ON informix.sysprobetables FROM public AS informix;
GRANT SELECT ON informix.sysprobetables TO public as informix;
```

The sysprobecolumns table

The schema for the sysprobecolumns table is:

```
CREATE TABLE informix.sysprobecolumns
(
  dbname char(128), { database name }
  sql_id int8,      { statement id in syssqltrace }
  tabid integer,    { table id }
  colno smallint   { column number }
);
REVOKE ALL ON informix.sysprobecolumns FROM public AS informix;
GRANT SELECT ON informix.sysprobecolumns TO public AS informix;
```

The sysprobejds table

The schema for the sysprobejds table is:

```
CREATE TABLE informix.sysprobejds
(
  dbname char(128), { database name }
  sql_id int8, { statement id in syssqltrace }
  jd integer, { join descriptor sequence number }
  ctabid integer, { child table id }
  ptabid integer, { parent table id }
  type char(1), { join type: inner (i), left outer (l) }
  uniq char(1) { parent table has unique index (y/n) }
);
REVOKE ALL ON informix.sysprobejds FROM public AS informix;
GRANT SELECT ON informix.sysprobejds TO public AS informix;
```

The sysprobejps table

The schema for the sysprobejps table:

```
CREATE TABLE informix.sysprobejps
(
  dbname char(128),  { database name }
  sql_id int8,      { statement id in syssqltrace }
  jd integer,        { join descriptor sequence number }
  jp integer,        { join predicate sequence number }
  ccolno smallint,  { child table column number }
  pcolno smallint   { parent table column number }
);
```

```
REVOKE ALL ON informix.sysprobejps FROM public AS informix;  
GRANT SELECT ON informix.sysprobejps TO public AS informix;
```

Related information

[Creating data mart definitions manually by using workload analysis on page 101](#)

Index

Special Characters

/dev/shm 83

A

- Accelerated query tables 52, 95
 - described 52, 95
 - states 95
- Accelerator
 - connecting to Informix 94
- accelerator server
 - directory 71
- Accelerator server
 - configuring 76
 - configuring for hardware clusters 77
- Access privileges 21
- Administration interface *see* Tools for administering
- Aggregate functions 67
- Aggregate tables 96
- ALTER FRAGMENT ADD CONSTRAINT statement 48
- ALTER FRAGMENT ATTACH statement 48
- ALTER FRAGMENT DETACH statement 48
- ALTER FRAGMENT statement 40
- ALTER TABLE MODIFY statement 48
- ALTER TABLE statement 46
- AQTs *see* Accelerated query tables
- Attached index 21
- AUS_CHANGE configuration parameter 48
- Auto Update Statistics (AUS) maintenance system 48
- AUTO_STAT_MODE configuration parameter 45, 48

B

- BSON
 - example 112
- business process
 - defined 13

C

- CLUSTER_INTERFACE parameter 77, 80
- cluster.conf file 77
- Configuring
 - accelerator server 76
 - dwainst.conf file 80
 - dwavp 72
 - overview 76
 - sbspace 72
 - SBSPACE_NAME configuration parameter 72
 - secondary servers 79
- Configuring hardware clusters
 - accelerator server 77
- conn.prop.std 60
- Coordinator node 58
 - memory issues 83
- COORDINATOR_SHM parameter 80
- CREATE DATABASE statement 34
 - dimensional data model 34
- CREATE INDEX statement 48
- CREATE TABLE statement 46
 - dimensional data model 34
- CREATE TABLE statements 34

D

- Data mart

- description 1
- Data mart definitions
 - creating 101
 - defined 99
- Data marts 99
 - aggregate tables 96
 - defined 99
 - described 52, 96
 - dimension table 96
 - dropping 118
 - example, creating 103
 - fact table 96
 - listing 91
 - loading data 109
 - overview 96
 - re-creating 116
 - reloading 115
 - replacing 117
 - star schema 96
 - summary tables 96
 - time series data 118
- data modeling
 - dimensional 1
 - normalized 4
- Data models
 - dimensional 8, 12
- Data types
 - supported 65
- Data warehouse
 - denormalization 1
 - description 1
- Databases
 - demonstration
 - sales_demo 13
- Detached index 21
- Dimension table
 - changing dimensions 31
 - choosing attributes 19
 - description 12
- Dimension tables
 - creating 34
 - sample 96
- dimensional data model
 - creating fact tables 34
- Dimensional data model
 - building 12
 - changing dimensions 31
 - creating dimension tables 34
 - denormalization 19
 - designing 7
 - dimension attributes 11
 - dimension elements 10
 - dimension tables 12
 - dimensions 10
 - fact table 9
 - implementing 34
 - measures, definition 9
 - minidimension tables 30
- Dimensional database 34, 34
 - loading data 38
 - loading from external tables 43
 - snowflake schema 32
 - testing 39
- dimensional database model 1
- Dimensional databases
 - mapping data sources 36
- Dimensional table
 - identifying granularity 16

- loading data 38
- Directories 71
 - accelerator server 71
 - accelerator server storage 76
 - documentation 71
 - installation 71
 - samples 71
 - storage 71
- Directories for hardware clusters
 - accelerator server storage 77
- Discretionary access privileges 21
- Distributed storage designs 21
- DRDA_INTERFACE parameter 80
- DWA_CM binary 88
- dwa_java_quickref.txt 60
- dwa_java_reference.txt 60
- dwa_message_reference.txt 60
- dwacli.jar 60
- DWADIR parameter 80
- dwainst.conf file
 - ondwa setup command 88
 - parameters 80
- dwavp
 - adding 72
 - configuring 72

E

- Examples
 - BSON 112
 - JSON acceleration 112
 - workload analysis 103
- EXPLAIN_STAT configuration parameter 45
- external tables
 - example 109
 - flat files 109
 - ifx_loadPartMart 109
 - joins 109
 - load data 109
 - pipes 109
 - support 109

F

- Fact table
 - description 8, 9
 - determining granularity 14
 - dimensions 16
 - granularity 9
- Fact tables
 - creating 34
 - sample 96
- Foreign key 18
- Fragment elimination 21
- Fragment expression 21
- Fragment key 21
- Fragment list 21
- Fragmentation key 18
- Fragmentation strategies
 - by expression 25
 - by interval 28
 - by list 27
 - by round-robin 24
- Functions
 - aggregate functions 67
 - ifx_createMart() 136
 - ifx_dropMart() 139
 - ifx_dropPartMart() 140
 - ifx_genMartDef() 141
 - ifx_getdwaMetrics() 142

- ifx_getMartdef() 145
 - ifx_getMartStat() 146
 - ifx_listMarts() 148
 - ifx_loadMart() 149
 - ifx_loadPartMart() 151
 - ifx_refreshMart() 154
 - ifx_removeDWA() 156
 - ifx_removeTrickleFeed() 156
 - ifx_setMart() 157
 - ifx_setupDWA() 158
 - ifx_setupTrickleFeed() 159
 - ifx_TSDW_setCalendar() 168
 - ifx_TSDW_updatePartition() 166
 - scalar functions 67
 - user-defined functions 67
- G**
- granularity
 - data distribution statistics 46
 - identifying dimensions 16
 - Granularity, fact table 9
- H**
- High-availability secondary servers
 - configuring 79
- I**
- I/O contention 21
 - ifx_createMart() function 136
 - ifx_def2Mart() procedure 138
 - ifx_dropMart() function 139
 - ifx_dropPartMart() function 140
 - ifx_genMartDef() function 141
 - ifx_getdwaMetrics() function 142
 - ifx_getMartdef() function 145
 - ifx_getMartStat() function 146
 - ifx_listMarts() function 148
 - ifx_loadMart() function 149
 - ifx_loadPartMart() function 151
 - ifx_probe2Mart() procedure 152
 - ifx_refreshMart() function 154
 - ifx_removeDWA() function 156
 - ifx_removeTrickleFeed() function 156
 - ifx_setMart() function 157
 - ifx_setupDWA() function 158
 - ifx_setupTrickleFeed() function 159
 - ifx_TSDW_createWindow() function 162
 - ifx_TSDW_dropWindow() function 163
 - ifx_TSDW_moveWindows() function 165
 - ifx_TSDW_setCalendar() function 168
 - ifx_TSDW_updatePartition() function 166
 - ifx_xml2Mart() procedure 169
 - Informix
 - Warehouse Accelerator
 - architecture 55
 - installing 74
 - verify setup 73
 - installing 70
 - Installing
 - directory 71
 - Informix
 - Warehouse Accelerator
 - 74
 - overview 70
 - verify server setup 73
- J**
- Java classes 60
 - Java commands 60
 - Join descriptors 107, 174
 - Join predicates 107, 174
 - Joins
 - join combinations 69
 - join predicates 69
 - supported 69
 - unsupported 69
- K**
- Key steps for accelerating queries 54
- L**
- list data marts
 - command 91
 - Logging table
 - creation 34
- M**
- Memory issues 83
- N**
- Nonlogging tables
 - creation 34
 - normalized database model 4
 - NULL fragment 27
 - NUM_NODES parameter 80
- O**
- OLAP window functions 64
 - ondwa utility
 - clean command 94
 - dwainst.conf file 88
 - getpin command 91
 - listmarts command 91
 - overview 86
 - reset command 93
 - running a cluster system 86
 - setup command 88
 - start command 89
 - status command 89
 - stop command 93
 - tasks command 92
 - users who can run the command 87
 - ondwack script 73
 - Online analytical processing (OLAP) 4
 - Online transaction processing (OLTP) 4
 - onstat -g aqt command 95
 - operating system 70
 - Operating systems
 - supported 70
 - Operational data store
 - description 1
 - Overview 52
- P**
- Parallel-database queries (PDQ) 21
 - Permissions for accelerator tools 59
 - Prerequisites
 - operating system 70
 - software 70
 - Primary key 18
 - Probing data
 - removing 108
 - Procedures
 - ifx_def2Mart() 138
 - ifx_probe2Mart() procedure 152
 - ifx_xml2Mart() 169
- Q**
- Queries
 - acceleration considerations 61
 - not accelerated 65
 - types accelerated 61
 - Queries blocks
 - types accelerated 61
 - query execution plans
 - considerations 45
 - query optimizer 45
 - Query probing
 - creating data marts 101
 - example 103
 - removing data 108
- R**
- Range fragments 28
 - Range interval distribution 28
 - Referential constraints
 - foreign key 18
 - primary key 18
 - reinstalling 75
 - Reinstalling 75
 - REMAINDER fragment 27
 - Routines
 - character limits 134
 - ifx_TSDW_createWindow() 162
 - ifx_TSDW_dropWindow() 163
 - ifx_TSDW_moveWindows() 165
- S**
- sales_demo database
 - creating 34
 - data model 13
 - data sources for 36
 - loading 38
 - Samples
 - query
 - profit by store 63
 - revenue by item 62
 - revenue by store 63
 - week to day profits 64
 - warehouse schema 171
 - sbspace
 - configuring 72
 - SBSPPACENAME configuration parameter 45, 72
 - Scalar functions 67
 - Schemas
 - sample SQL 171
 - Secondary servers
 - configuring 79
 - SET ENVIRONMENT
 - statement 108
 - SET ENVIRONMENT AUTO_STAT_MODE
 - statement 48
 - SET ENVIRONMENT STATCHANGE
 - statement 46, 48
 - SET ENVIRONMENT USE_DWA
 - statement of SQL 129
 - Shared memory configuration 83
 - SHMMAX kernel parameter 83
 - Snowflake schema 52
 - example 32
 - Star schema
 - description 8
 - sample 96
 - START_PORT parameter 80
 - STATCHANGE configuration parameter 45, 46, 48
 - STATCHANGE table attribute 48
 - STATCHANGE table property 46
 - STATLEVEL table property 46
 - Summary tables 96
 - Swap space 83
 - Swap space configuration 83
 - SYSDISTRIB system catalog table 45

SYSFRAGDIST system catalog table 45, 46
SYSFRAGMENTS system catalog table 45
SYSINDICES system catalog table 45
SYSSBSPACENAME configuration parameter 46

T

tables
 external tables 109
 JSON 111
 NoSQL 111
 synonyms 111
 time series virtual tables 112
 views 111
time series
 calendar 119
 calendar index 120
 continuous refresh 126
 create time window 120
 drop data mart 122
 example: ifx_TSDW_createWindow 122
 example: ifx_TSDW_dropWindow 122
 example: ifx_TSDW_moveWindows 122
 example: ifx_TSDW_setCalendar 122
 example: ifx_TSDW_updatePartition 122
 ifx_TSDW_createWindow 120
 ifx_TSDW_dropWindow 120
 ifx_TSDW_moveWindows 120
 ifx_TSDW_setCalendar 119
 ifx_TSDW_updatePartition 122
 move time window 120, 122
 refresh procedure 126
 remove time window 120, 122
 stores_demo 122
 time windows 120
 virtual partition 119, 120
 virtual partition refresh 122
time series data
 calendar 118
 refresh 118
 time window 118
 virtual partition 118
time window
 create 120
 move 120
 remove 120
Transition fragment 21
Transition value 21
Troubleshooting
 memory issues 83
 sbspace 170

U

uninstalling 75
Uninstalling 75
UNION ALL set operator 64
UNION set operator 64
Unique constraints 18
UPDATE STATISTICS statement 48
use_dwa session environment variable 129, 130
use_dwa variable environment variable 170
User-defined functions 67
Utility program
 dbload 34

V

Virtual memory usage 83
virtual partition
 identify 120
 refresh 122
vm.overcommit_memory kernel parameter 83

vm.overcommit_ratio kernel parameter 83

W

WAREHOUSE privilege 59
Worker nodes 58
 memory issues 83
WORKER_SHM parameter 80
Workload analysis 100
 creating data marts 101