

Introduction

- 1. Purpose of Database Systems**
- 2. View of Data**
- 3. Data Models**
- 4. Data Definition Language**
- 5. Data Manipulation Language**
- 6. Transaction Management**
- 7. Storage Management**
- 8. Database Administrator**
- 9. Database Users**
- 10. Overall System Structure**

Well we can define following constraints in DBMS

Primary Key: Its main features is that it must contain a unique value for each row of data. It cannot contain null values.

Foreign Key: Its a field in one table that uniquely identifies a row of another table or the same table. In other words, the foreign key is defined in a second table, but it refers to the primary key or a unique **key** in the first table

Not Null: This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field

Unique Key: Its ensures that all values in a column are different. Both the UNIQUE and PRIMARY **KEY constraints** provide a guarantee for uniqueness for a column or set of columns

Check: Its a check constraint is a type of integrity constraint in SQL which specifies a requirement that must be met by each row in a database table.


Domain Constraints: These are user defined data type and we can define them like this: Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT). In other words we can say that Domain Constrains is user defined constraint to apply group constraints.

Primary Key Constraints

A table typically has a column or combination of columns that contain values that uniquely identify each row in the table. This column, or columns, is called the primary key (PK) of the table and enforces the entity integrity of the table. Because primary key constraints guarantee unique data, they are frequently defined on an identity column. When you specify a primary key constraint for a table, the Database Engine enforces data uniqueness by automatically creating a unique index for the primary key columns.

This index also permits fast access to data when the primary key is used in queries. If a primary key constraint is defined on more than one column, values may be duplicated within one column, but each combination of values from all the columns in the primary key constraint definition must be unique.

As shown in the following illustration, the **ProductID** and **VendorID** columns in the **Purchasing.ProductVendor** table form a composite primary key constraint for this table. This makes sure that every row in the **ProductVendor** table has a unique combination of **ProductID** and **VendorID**. This prevents the insertion of duplicate rows.



ProductID	VendorID	AverageLeadTime	StandardPrice	LastReceiptCost
1	1	17	47.8700	50.2635
2	104	19	39.9200	41.9160
7	4	17	54.3100	57.0255
609	7	17	25.7700	27.0585
609	100	19	28.1700	29.5785

ProductVendor table

A table can contain only one primary key constraint.

A primary key cannot exceed 16 columns and a total key length of 900 bytes.

The index generated by a primary key constraint cannot cause the number of indexes on the table to exceed 999 nonclustered indexes and 1 clustered index.

If clustered or nonclustered is not specified for a primary key constraint, clustered is used if there no clustered index on the table.

All columns defined within a primary key constraint must be defined as not null. If nullability is not specified, all columns participating in a primary key constraint have their nullability set to not null.

If a primary key is defined on a CLR user-defined type column, the implementation of the type must support binary ordering.

Foreign Key Constraints

A foreign key (FK) is a column or combination of columns that is used to establish and enforce a link between the data in two tables to control the data that can be stored in the foreign key table. In a foreign key reference, a link is created between two tables when the column or columns that hold the primary key value for one table are referenced by the column or columns in another table. This column becomes a foreign key in the second table. For example, the **Sales.SalesOrderHeader** table has a foreign key link to the **Sales.SalesPerson** table because there is a logical relationship between sales orders and salespeople. The **SalesPersonID** column in the **SalesOrderHeader** table matches the primary key column of the **SalesPerson** table. The **SalesPersonID** column in the **SalesOrderHeader** table is the foreign key to the **SalesPerson** table. By creating this foreign key relationship, a value for **SalesPersonID** cannot be inserted into the **SalesOrderHeader** table if it does not already exist in the **SalesPerson** table.

A table can reference a maximum of 253 other tables and columns as foreign keys (outgoing references). SQL Server 2016 increases the limit for the number of other table and columns that can reference columns in a single table (incoming references), from 253 to 10,000. (Requires at least 130 compatibility level.) The increase has the following restrictions:

Greater than 253 foreign key references are only supported for DELETE DML operations. UPDATE and MERGE operations are not supported.

A table with a foreign key reference to itself is still limited to 253 foreign key references.

Greater than 253 foreign key references are not currently available for columnstore indexes, memory-optimized tables, Stretch Database, or partitioned foreign key tables.

Indexes on Foreign Key Constraints

Unlike primary key constraints, creating a foreign key constraint does not automatically create a corresponding index. However, manually creating an index on a foreign key is often useful for the following reasons:

Foreign key columns are frequently used in join criteria when the data from related tables is combined in queries by matching the column or columns in the foreign key constraint of one table with the primary or unique key column or columns in the other table. An index enables the Database Engine to quickly find related data in the foreign key table. However, creating this index is not required. Data from two related tables can be combined even if no primary key or foreign key constraints are defined between the tables, but a foreign key relationship between two tables indicates that the two tables have been optimized to be combined in a query that uses the keys as its criteria.

Changes to primary key constraints are checked with foreign key constraints in related tables.

Referential Integrity

Although the main purpose of a foreign key constraint is to control the data that can be stored in the foreign key table, it also controls changes to data in the primary key table. For example, if the row for a salesperson is deleted from the **Sales.SalesPerson** table, and the salesperson's ID is used for sales orders in the **Sales.SalesOrderHeader** table, the relational integrity between the two tables is broken; the deleted salesperson's sales orders are orphaned in the **SalesOrderHeader** table without a link to the data in the **SalesPerson** table. A foreign key constraint prevents this situation. The constraint enforces referential integrity by guaranteeing that changes cannot be made to data in the primary key table if those changes invalidate the link to data in the foreign key table. If an attempt is made to delete the row in a primary key table or to change a primary key value, the action will fail when the deleted or changed primary key value corresponds to a value in the foreign key constraint of another table. To successfully change or delete a row in a foreign key constraint, you must first either delete the foreign key data in the foreign key table or change the foreign key data in the foreign key table, which links the foreign key to different primary key data.

Cascading Referential Integrity

By using cascading referential integrity constraints, you can define the actions that the Database Engine takes when a user tries to delete or update a key to which existing foreign keys point. The following cascading actions can be defined.

NO ACTION

The Database Engine raises an error and the delete or update action on the row in the parent table is rolled back.

CASCADE

Corresponding rows are updated or deleted in the referencing table when that row is updated or deleted in the parent table. CASCADE cannot be specified if a **timestamp** column is part of either the foreign key or the referenced key. ON DELETE CASCADE cannot be specified for a table that has an INSTEAD OF DELETE trigger. ON UPDATE CASCADE cannot be specified for tables that have INSTEAD OF UPDATE triggers.

SET NULL

All the values that make up the foreign key are set to NULL when the corresponding row in the parent table is updated or deleted. For this constraint to execute, the foreign key columns must be nullable. Cannot be specified for tables that have INSTEAD OF UPDATE triggers.

SET DEFAULT

All the values that make up the foreign key are set to their default values if the corresponding row in the parent table is updated or deleted. For this constraint to execute, all foreign key columns must have default definitions. If a column is nullable, and there is no explicit default value set, NULL becomes the implicit default value of the column. Cannot be specified for tables that have INSTEAD OF UPDATE triggers. CASCADE, SET NULL, SET DEFAULT and NO ACTION can be combined on tables that have referential relationships with each other. If the Database Engine encounters NO ACTION, it stops and rolls back related CASCADE, SET NULL and SET DEFAULT actions. When a DELETE statement causes a combination of CASCADE, SET NULL, SET DEFAULT and NO ACTION actions, all the CASCADE, SET NULL and SET DEFAULT actions are applied before the Database Engine checks for any NO ACTION.

Database Users

- Users are differentiated by the way they expect to interact with the system
- Application programmers – interact with system through DML calls
- Sophisticated users – form requests in a database query language
- Specialized users – write specialized database applications that do not fit into the traditional data processing framework
- Naïve users – invoke one of the permanent application programs that have been written previously
 - E.g. people accessing database over the web, bank tellers, clerical staff

Database Administrator

- Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties include:
 - Schema definition
 - Storage structure and access method definition
 - Schema and physical organization modification
 - Granting user authority to access the database
 - Specifying integrity constraints
 - Acting as liaison with users
 - Monitoring performance and responding to changes in requirements

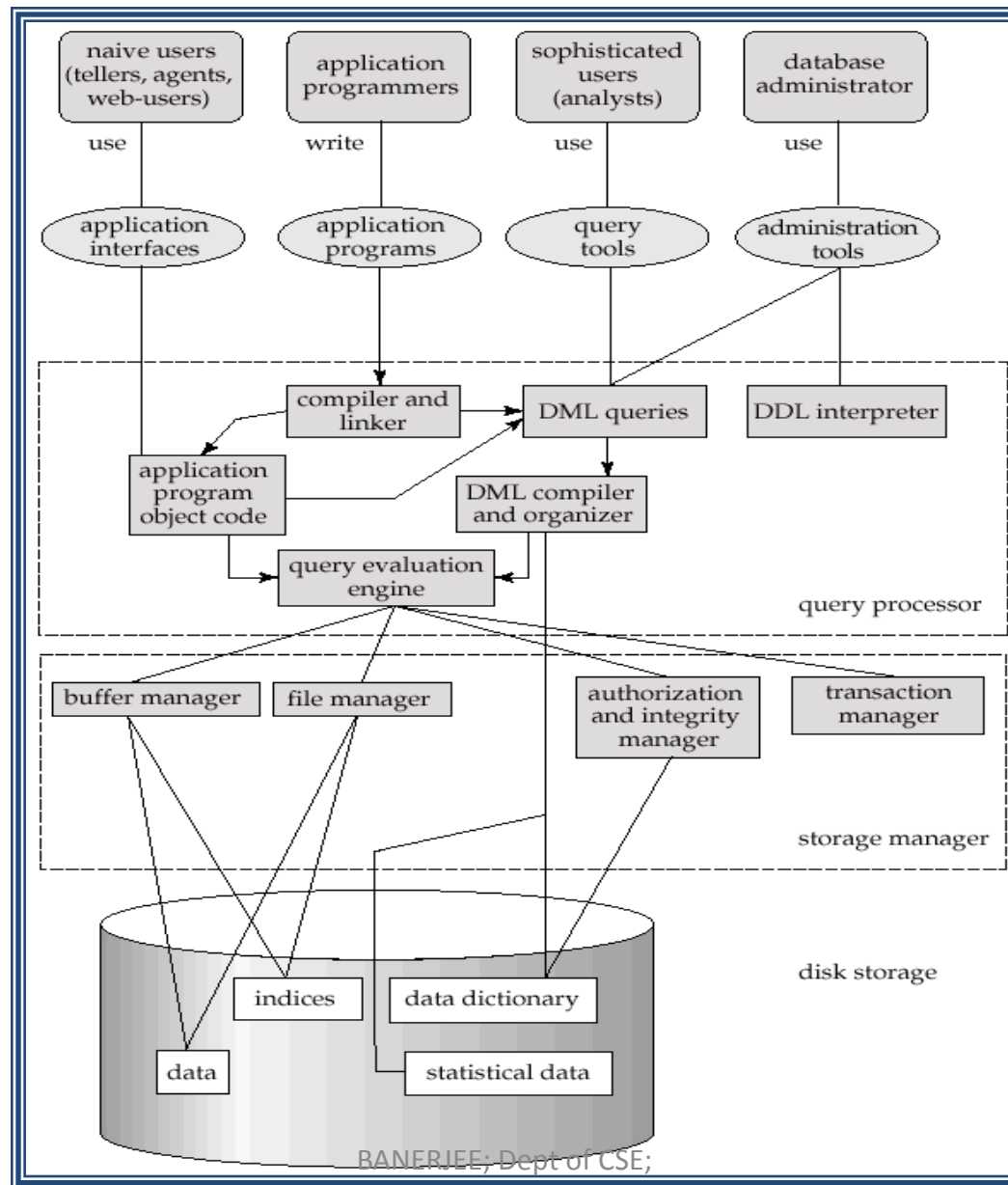
Transaction Management

- A *transaction* is a collection of operations that performs a single logical function in a database application
- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- Concurrency-control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

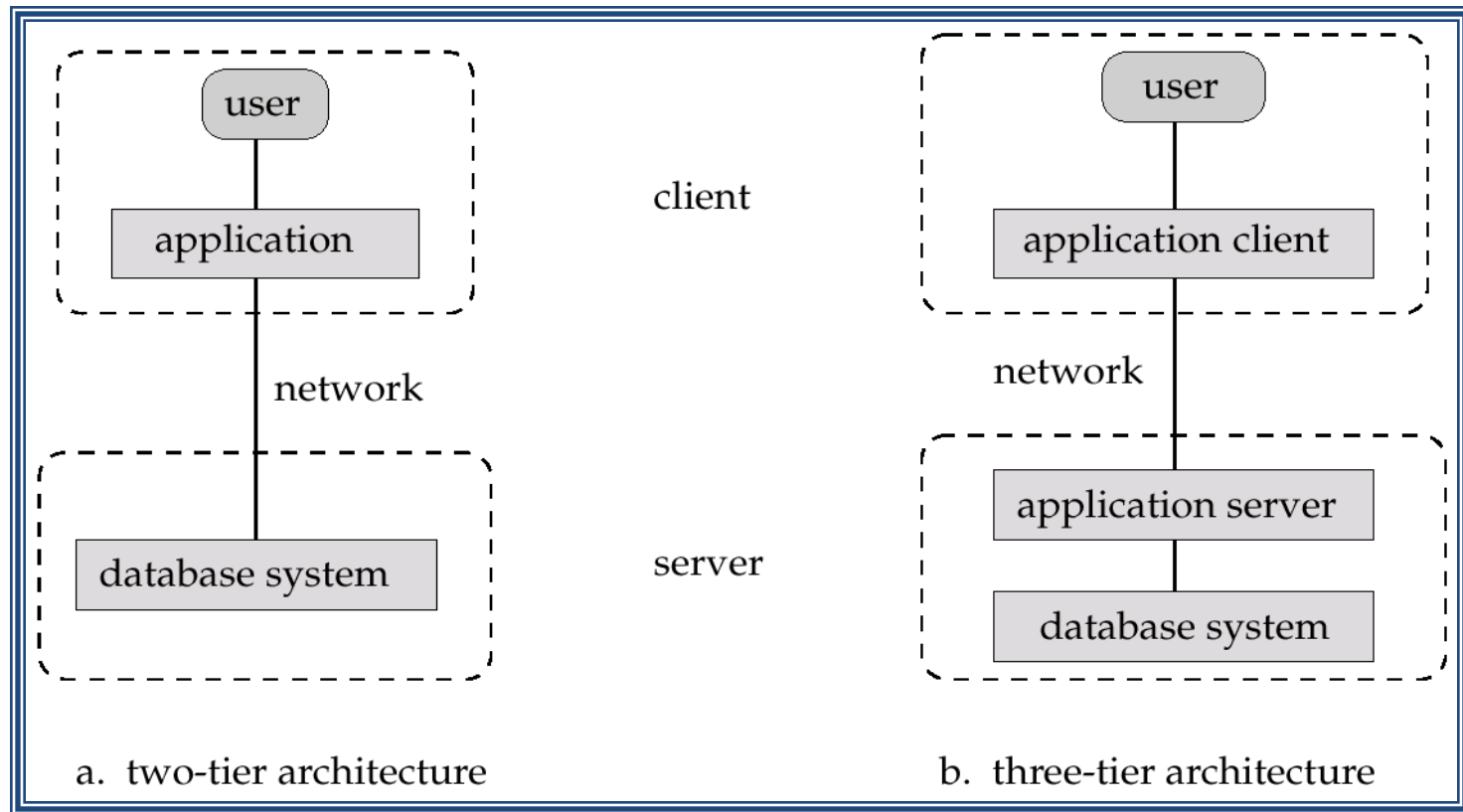
Storage Management

- Storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible to the following tasks:
 - interaction with the file manager
 - efficient storing, retrieving and updating of data

Overall System Structure



Application Architectures



- **Two-tier architecture:** E.g. client programs using ODBC/JDBC to communicate with a database
- **Three-tier architecture:** E.g. web-based applications, and applications built using “middleware”

An **application server** is a component-based product that resides in the middle-tier of a **server** centric architecture.

It provides middleware services for security and state maintenance, along with data access and persistence.

Unstructured data

In 1998, Merrill Lynch cited a rule of thumb that somewhere around 80-90% of all potentially usable business information may originate in unstructured form

1. Unstructured data (or unstructured information) refers to information that either does not have a pre-defined data model or is not organized in a pre-defined manner.
2. Unstructured information is typically text-heavy, but may contain data such as dates, numbers, and facts as well.
3. This results in irregularities and ambiguities that make it difficult to understand using traditional programs as compared to data stored in fielded form in databases or annotated (semantically tagged) in documents.

NoSQL Databases:

➤ **Diversity in semi structured and unstructured data**

➤ **Oracle NoSQL Database**

Column: Accumulo, Cassandra, Druid, HBase, Vertica, SAP HANA

Document: Apache CouchDB, ArangoDB, Clusterpoint, Couchbase, DocumentDB, HyperDex, IBM Domino, MarkLogic, **MongoDB**, OrientDB, Qizx, RethinkDB

Key-value: Aerospike, ArangoDB, Couchbase, Dynamo, FairCom c-treeACE, FoundationDB, HyperDex, InfinityDB, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, Berkeley DB, SDBM/Flat File dbm

Graph: AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso

Multi-model: Alchemy Database, ArangoDB, CortexDB, Couchbase, FoundationDB, InfinityDB, MarkLogic, OrientDB

Mongo DB

1. MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time
2. The document model maps to the objects in your application code, making data easy to work with
3. Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyze data
4. MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use
5. MongoDB is free and open-source, published under the GNU Affero General Public License

mySQL

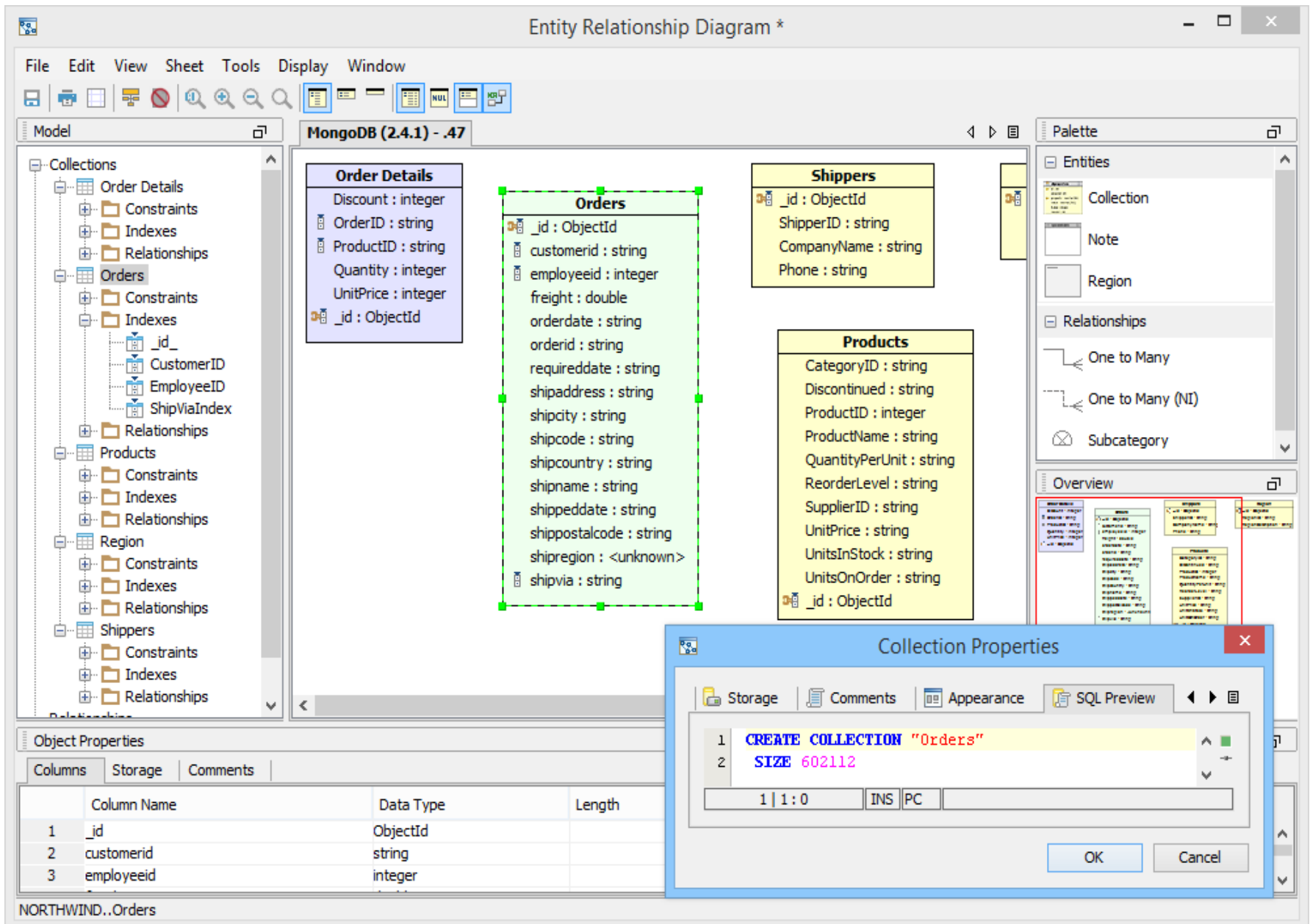
SELECT

```
Dim1, Dim2,  
SUM(Measure1) AS MSum,  
COUNT(*) AS RecordCount,  
AVG(Measure2) AS MAvg,  
MIN(Measure1) AS MMin  
MAX(CASE  
  WHEN Measure2 < 100  
  THEN Measure2  
END) AS MMax  
FROM DenormAggTable  
WHERE (Filter1 IN ('A','B'))  
AND (Filter2 = 'C')  
AND (Filter3 > 123)  
GROUP BY Dim1, Dim2  
HAVING (MMin > 0)  
ORDER BY RecordCount DESC  
LIMIT 4, 8
```

- ① Grouped dimension columns are pulled out as keys in the map function, reducing the size of the working set.
- ② Measures must be manually aggregated.
- ③ Aggregates depending on record counts must wait until finalization.
- ④ Measures can use procedural logic.
- ⑤ Filters have an ORM/ActiveRecord-looking style.
- ⑥ Aggregate filtering must be applied to the result set, not in the map/reduce.
- ⑦ Ascending: 1; Descending: -1

MongoDB

```
db.runCommand({  
  mapreduce: "DenormAggCollection",  
  query: {  
    filter1: { '$in': [ 'A', 'B' ] },  
    filter2: 'C',  
    filter3: { '$gt': 123 }  
  },  
  map: function() { emit(  
    { d1: this.Dim1, d2: this.Dim2 },  
    { msum: this.measure1, recs: 1, mmin: this.measure1,  
      mmax: this.measure2 < 100 ? this.measure2 : 0 }  
  ); },  
  reduce: function(key, vals) {  
    var ret = { msum: 0, recs: 0, mmin: 0, mmax: 0 };  
    for(var i = 0; i < vals.length; i++) {  
      ret.msum += vals[i].msum;  
      ret.recs += vals[i].recs;  
      if(vals[i].mmin < ret.mmin) ret.mmin = vals[i].mmin;  
      if((vals[i].mmax < 100) && (vals[i].mmax > ret.mmax))  
        ret.mmax = vals[i].mmax;  
    }  
    return ret;  
  },  
  finalize: function(key, val) {  
    val.mavg = val.msum / val.recs;  
    return val;  
  },  
  out: 'result1',  
  verbose: true  
});  
db.result1.  
  find({ mmin: { '$gt': 0 } }).  
  sort({ recs: -1 }).  
  skip(4).  
  limit(8);
```



Cassandra

1. The Apache Cassandra database is the right choice when you need scalability and high availability without compromising performance.
2. Linear scalability and proven fault-tolerance on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data.
3. Cassandra's support for replicating across multiple datacenters is best-in-class, providing lower latency for users and the peace of mind of knowing that you can survive regional outages.
4. Query language Cassandra introduced the Cassandra Query Language (CQL). CQL is a simple interface for accessing Cassandra, as an alternative to the traditional Structured Query Language (SQL).
5. CQL adds an abstraction layer that hides implementation details of this structure and provides native syntaxes for collections and other common encodings


```
CREATE KEYSPACE MyKeySpace
  WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 3 };

USE MyKeySpace;

CREATE COLUMNFAMILY MyColumns (id text, Last text, First text, PRIMARY KEY(id));

INSERT INTO MyColumns (id, Last, First) VALUES ('1', 'Doe', 'John');

SELECT * FROM MyColumns;
```

Which gives:

```
id | first | last
---+-----+-----
 1 |  John |  Doe

(1 rows)
```

Neo4j

1. **Neo4j** is a graph database management system developed by Neo Technology, Inc.
2. Described by its developers as an ACID-compliant transactional database with native graph storage and processing

Organizations using Neo4j



Labeled Property Graph Data Model

