|  |
| --- |
| EPAM Systems, RD Dep., RD Dep. |
| MTN.NIX.07 Oracle SQL Introduction  **Oracle SQL Basics** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial Version | Siarhei Kandrashevich | 15-Jan-2014 |  |  |
|  |  |  |  |  |  |

Contents

[1. The Evolution of Database Modeling 3](#_Toc380362864)

[1.1. The History of Relational Databases 5](#_Toc380362865)

[2. What is a relational database? 7](#_Toc380362866)

[3. Normalized data 11](#_Toc380362867)

[4. Primary and foreign keys 21](#_Toc380362868)

[5. ANSI and Oracle SQL 24](#_Toc380362869)

[2.1. SQL Standards 24](#_Toc380362870)

[2.2. Common Language for All Relational Databases 24](#_Toc380362871)

[2.3. ANSI Standard Structure 25](#_Toc380362872)

[2.4. Oracle Compliance To Core SQL:2008 26](#_Toc380362873)

[2.5. Character Sets Support 26](#_Toc380362874)

[6. Oracle SQL 26](#_Toc380362875)

[6.1. Basic Language Elements 26](#_Toc380362876)

[6.2. Oracle Database Objects Naming Rules (simplified) 27](#_Toc380362877)

[6.3. Oracle support for SELECT statement clauses 27](#_Toc380362878)

[6.4. Dummy table (DUAL) 27](#_Toc380362879)

[6.5. Set Operations 28](#_Toc380362880)

[7. Pseudocolumns 28](#_Toc380362881)

[7.1. ROWNUM Pseudocolumn 29](#_Toc380362882)

[7.2. ROWID Pseudocolumn 29](#_Toc380362883)

[8. Client Tools Support 30](#_Toc380362884)

[8.1. Client Tools Overview 30](#_Toc380362885)

[8.2. Oracle SQL Developer 31](#_Toc380362886)

[8.2.1. Download and Install 31](#_Toc380362887)

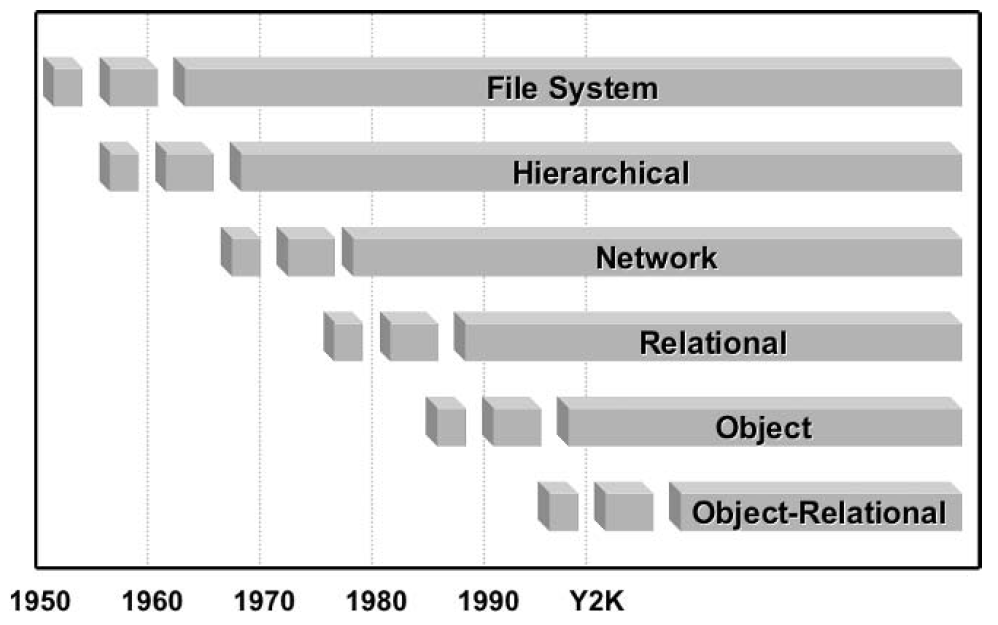
[8.2.2. Establish Connection 31](#_Toc380362888)

[8.2.3. Oracle Data Modeler 31](#_Toc380362889)

[9. External Sources 32](#_Toc380362890)

# The Evolution of Database Modeling

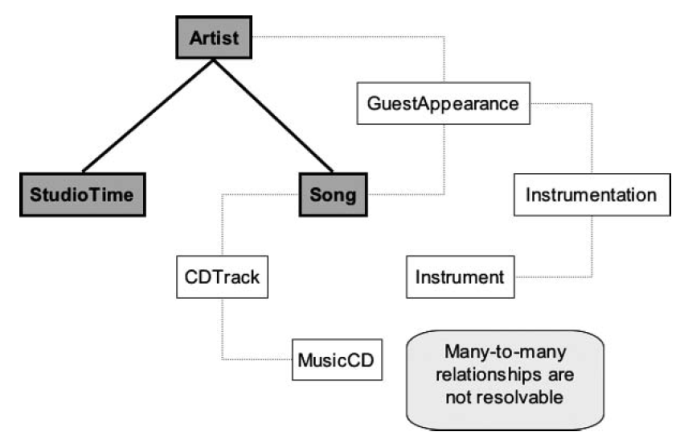
The history of databases is essentially the history of different data modeling techniques. Data modeling techniques have evolved over the last 50 years from use of simple file systems to relational, object, and object-relational models. Figure 1.1 shows the evolution of data modeling techniques.



**Figure 1.1** *The Evolution of Data Modeling.*

**Hierarchical**

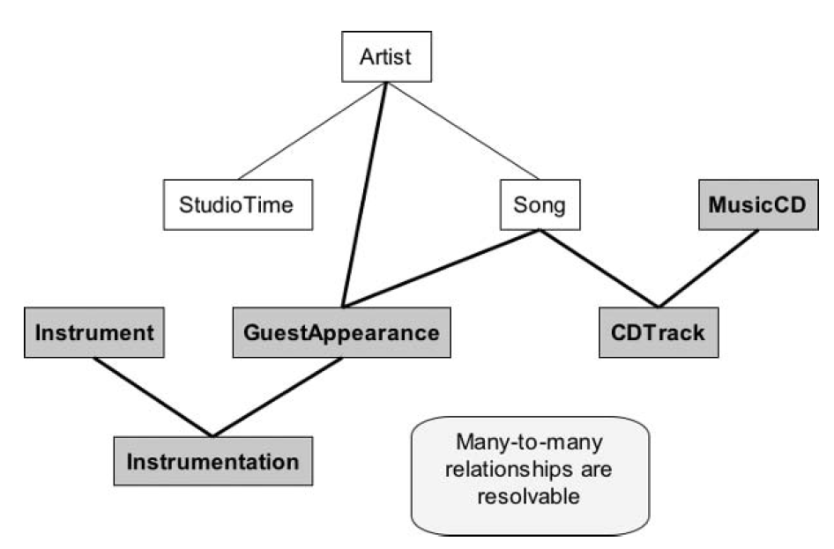
A branch-leaf tree structure as shown in Figure 1.2 such that child tables can only have single parent tables. A child table is completely dependent on the existence of its parent table. As a result, one-to-many relationships are supported but not many-to many relationships. The primary disadvantage of a hierarchical structure is that everything must be accessed from the root node of the tree. In Figure 1.2, accessing a Song would require retrieval of an Artist and all of that artist’s songs.



**Figure 1.2** *The Hierarchical Data Model.*

* **Network**

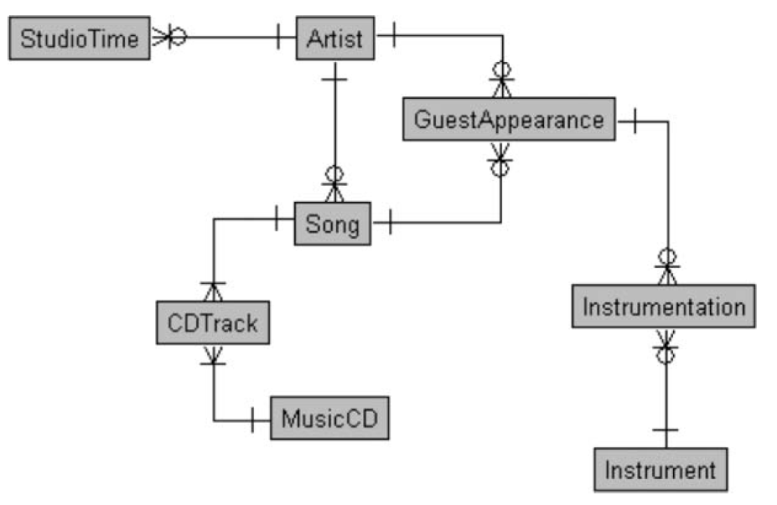
Refinement of the hierarchical model where many-to many relationships are permitted because child tables can have more than a single parent table. This creates a “networked” structure of tables as shown in Figure 1.3. A network structure is an improved hierarchical or branch-leaf tree structure where many-to-many entities can be accessed, but access to a node still requires access to all parent nodes from the root node.



**Figure 1.3** *The Network Data Model.*

* **Relational**

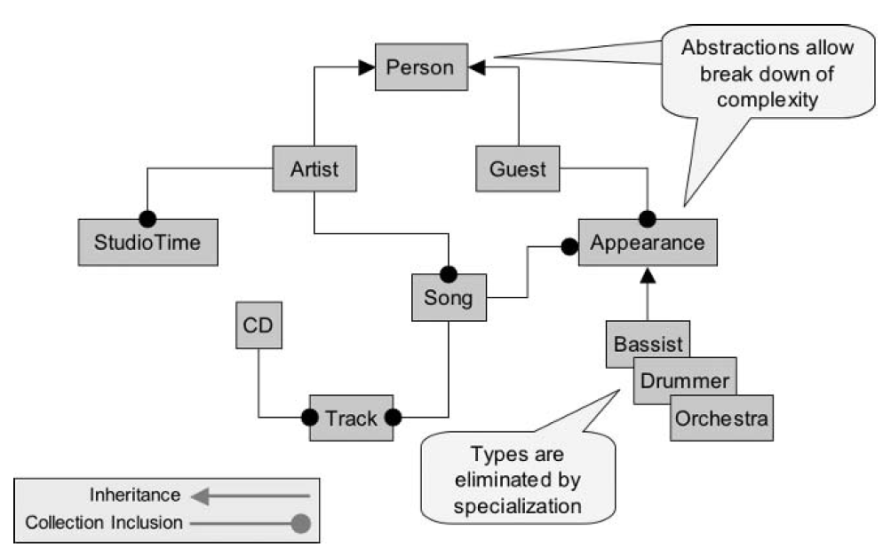
Any two tables can be linked irrespective of hierarchical placement. Therefore, any table can be accessed directly without having to access child tables through a hierarchy or network of parent tables. Relatively complex and efficient data structures can be created with the relational data model. The operative phrase for use of relational tables is rapid selection of groups of data rather than single items. Relational databases are most effective for reporting. An example relational structure is shown in Figure 1.4 where any table can be retrieved from based on key values. Tables or entities are built from those keys.



**Figure 1.4** *The Relational Data Model.*

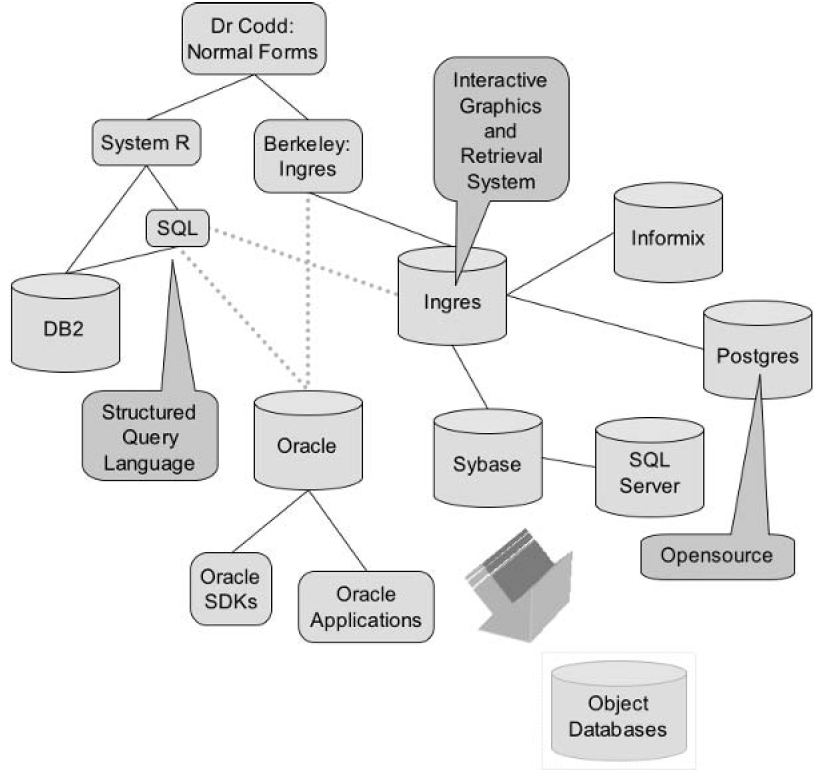
Object-Relational

Without losing efficiency, minimal object capabilities can be included in a relational data model. Be warned that relational and object data modeling is completely contrary, and “building too many” objects in a relational database will likely result in serious impact on general application performance. It is usually best to build relations in a relational database and reserve complex object structure for application code. Figure 1.6 shows storing of binary images into a relational database.



**Figure 1.5** *The Object Data Model.*

## The History of Relational Databases

Relational databases began with several papers written by Dr. Edgar F. Codd. Numerous other papers followed by various other researchers. Figure 1.7 shows

**Figure 1.6** *Including Multimedia in a Relational Database.*

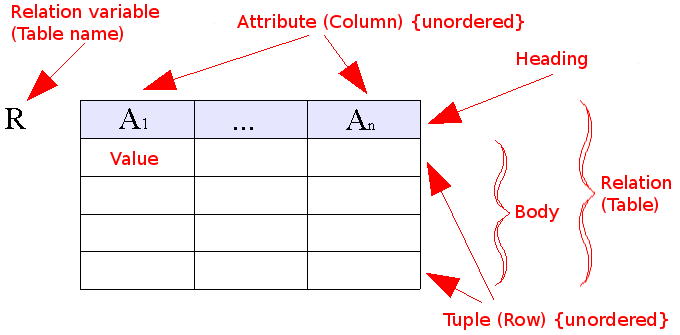
several distinct branches of development. These branches were DB2 from IBM, Oracle Database from Oracle Corporation, and a multitude of relational databases stemming from Ingres, which was initially conceived by two scientists at the University of California at Berkeley. In Figure 1.7, the most important point to note about the general development path of relational databases is as follows: Development from one database to another resided usually in different companies and was characterized by movement of personnel rather than of database source code. In other words, the people invented the different databases, not the companies, where people moved between different companies. Additionally, numerous object databases have been developed. Object databases generally have distinct applications. Some object databases have their roots in relational technology, once again in terms of the movement of personnel skills.

# What is a relational database?

The relational model for database management is a database model based on first-order predicate logic, first formulated and proposed in 1969 by Edgar F. Codd.

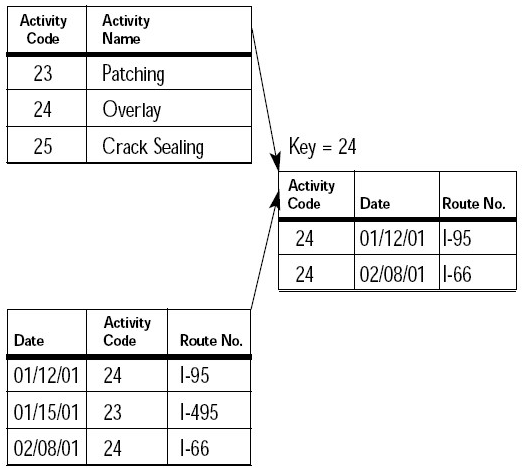
Diagram of an example database according to the Relational model.

In the relational model, related records are linked together with a "key".



**Figure 2.1** *Relational model concepts*

The purpose of the relational model is to provide a declarative method for specifying data and queries: users directly state what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries.



**Figure 2.2** *Relational model*

**The model**

The fundamental assumption of the relational model is that all data is represented as mathematical n-ary relations, an n-ary relation being a subset of the Cartesian product of n domains. In the mathematical model, reasoning about such data is done in two-valued predicate logic, meaning there are two possible evaluations for each proposition: either true or false (and in particular no third value such as unknown, or not applicable, either of which are often associated with the concept of NULL). Some think two-valued logic is an important part of the relational model, while others think a system that uses a form of three-valued logic can still be considered relational. Data are operated upon by means of a relational calculus or relational algebra, these being equivalent in expressive power.

The relational model of data permits the database designer to create a consistent, logical representation of information. Consistency is achieved by including declared constraints in the database design, which is usually referred to as the logical schema. The theory includes a process of database normalization whereby a design with certain desirable properties can be selected from a set of logically equivalent alternatives. The access plans and other implementation and operation details are handled by the DBMS engine, and are not reflected in the logical model. This contrasts with common practice for SQL DBMSs in which performance tuning often requires changes to the logical model.

The basic relational building block is the domain or data type, usually abbreviated nowadays to type. A tuple is an ordered set of attribute values. An attribute is an ordered pair of attribute name and type name. An attribute value is a specific valid value for the type of the attribute. This can be either a scalar value or a more complex type.

A relation consists of a heading and a body. A heading is a set of attributes. A body (of an n-ary relation) is a set of n-tuples. The heading of the relation is also the heading of each of its tuples.

A relation is defined as a set of n-tuples. In both mathematics and the relational database model, a set is an unordered collection of unique, non-duplicated items, although some DBMSs impose an order to their data. In mathematics, a tuple has an order, and allows for duplication. E.F. Codd originally defined tuples using this mathematical definition. Later, it was one of E.F. Codd's great insights that using attribute names instead of an ordering would be so much more convenient (in general) in a computer language based on relations. This insight is still being used today. Though the concept has changed, the name "tuple" has not. An immediate and important consequence of this distinguishing feature is that in the relational model the Cartesian product becomes commutative.

A table is an accepted visual representation of a relation; a tuple is similar to the concept of row, but note that in the database language SQL the columns and the rows of a table are ordered. A relvar is a named variable of some specific relation type, to which at all times some relation of that type is assigned, though the relation may contain zero tuples.

The basic principle of the relational model is the Information Principle: all information is represented by data values in relations. In accordance with this Principle, a relational database is a set of relvars and the result of every query is presented as a relation.

The consistency of a relational database is enforced, not by rules built into the applications that use it, but rather by constraints, declared as part of the logical schema and enforced by the DBMS for all applications. In general, constraints are expressed using relational comparison operators, of which just one, "is subset of" (?), is theoretically sufficient. In practice, several useful shorthands are expected to be available, of which the most important are candidate key (really, superkey) and foreign key constraints.

**Interpretation**

To fully appreciate the relational model of data it is essential to understand the intended interpretation of a relation.

The body of a relation is sometimes called its extension. This is because it is to be interpreted as a representation of the extension of some predicate, this being the set of true propositions that can be formed by replacing each free variable in that predicate by a name (a term that designates something).

There is a one-to-one correspondence between the free variables of the predicate and the attribute names of the relation heading. Each tuple of the relation body provides attribute values to instantiate the predicate by substituting each of its free variables. The result is a proposition that is deemed, on account of the appearance of the tuple in the relation body, to be true. Contrariwise, every tuple whose heading conforms to that of the relation but which does not appear in the body is deemed to be false. This assumption is known as the closed world assumption: it is often violated in practical databases, where the absence of a tuple might mean that the truth of the corresponding proposition is unknown. For example, the absence of the tuple ('John', 'Spanish') from a table of language skills cannot necessarily be taken as evidence that John does not speak Spanish.

For a formal exposition of these ideas, see the section Set-theoretic Formulation, below.

**Application to databases**

A data type as used in a typical relational database might be the set of integers, the set of character strings, the set of dates, or the two boolean values true and false, and so on. The corresponding type names for these types might be the strings "int", "char", "date", "boolean", etc. It is important to understand, though, that relational theory does not dictate what types are to be supported; indeed, nowadays provisions are expected to be available for user-defined types in addition to the built-in ones provided by the system.

Attribute is the term used in the theory for what is commonly referred to as a column. Similarly, table is commonly used in place of the theoretical term relation (though in SQL the term is by no means synonymous with relation). A table data structure is specified as a list of column definitions, each of which specifies a unique column name and the type of the values that are permitted for that column. An attribute value is the entry in a specific column and row, such as "John Doe" or "35".

A tuple is basically the same thing as a row, except in an SQL DBMS, where the column values in a row are ordered. (Tuples are not ordered; instead, each attribute value is identified solely by the attribute name and never by its ordinal position within the tuple.) An attribute name might be "name" or "age".

A relation is a table structure definition (a set of column definitions) along with the data appearing in that structure. The structure definition is the heading and the data appearing in it is the body, a set of rows. A database relvar (relation variable) is commonly known as a base table. The heading of its assigned value at any time is as specified in the table declaration and its body is that most recently assigned to it by invoking some update operator (typically, INSERT, UPDATE, or DELETE). The heading and body of the table resulting from evaluation of some query are determined by the definitions of the operators used in the expression of that query. (Note that in SQL the heading is not always a set of column definitions as described above, because it is possible for a column to have no name and also for two or more columns to have the same name. Also, the body is not always a set of rows because in SQL it is possible for the same row to appear more than once in the same body.)

# Normalized data

**What Is Normalization?**

At its most basic level, normalization is the process of simplifying your data into its most efficient form by eliminating redundant data. Understanding the definition of the word efficient in relation to normalization is the key concept. Efficiency, in this case, refers to reducing complexity from a logical standpoint. Efficiency does not necessarily equal better performance, nor does it necessarily equate to efficient query processing. This may seem to contradict what you've heard about design, so first let's walk through the concepts in normalization, and then we'll talk about some of the performance considerations.

**Normal Forms**

E. F. Codd, who was the IBM researcher credited with the creation and evolution of the relational database, set forth a set of rules that define how data should be organized in a relational database. Initially, he proposed three sequential forms to classify data in a database: first normal form (1NF), second normal form (2NF), and third normal form (3NF). After these initial normal forms were developed, research indicated that they could result in update anomalies, so three additional forms were developed to deal with these issues: fourth normal form (4NF), fifth normal form (5NF), and the Boyce-Codd normal form (BCNF). There has been research into a sixth normal form (6NF); this normal form has to do with temporal databases and is outside the scope of this book.

| **Artist Name** | **Genre** | **Album Name** | **Album Release Date** |
| --- | --- | --- | --- |
| The Awkward Stage | Rock | Home | 10/01/2006 |
| Girth | Metal | On the Sea | 5/25/1997 |
| Wasabi Peanuts | Adult Contemporary Rock | Spicy Legumes | 11/12/2005 |
| The Bobby | R&B | Live! | 7/27/1985 |
| Jenkins Band |  | Running the Game | 10/30/1988 |
| Juices of Brazil | Latin Jazz | Long Road | 1/01/2003 |
|  |  | White | 6/10/2005 |

**Table 3.1.** *Artists and Albums: Repeating Groups of Data*

It's important to note that the normal forms are nested. For example, if a database meets 3NF, by definition it also meets 1NF and 2NF. Let's take a brief look at each of the normal forms and explain how to identify them.

First Normal Form (1NF)

In first normal form, every entity in the database has a primary key attribute (or set of attributes). Each attribute must have only one value, and not a set of values. For a database to be in 1NF it must not have any repeating groups. A repeating group is data in which a single instance may have multiple values for a given attribute.

For example, consider a recording studio that stores data about all its artists and their albums. Table 3.1 outlines an entity that stores some basic data about the artists signed to the recording studio.

Notice that for the first artist, there is only one album and therefore one release date. However, for the fourth and fifth artists, there are two albums and two release dates. In practice, we cannot guarantee which release date belongs to which album. Sure, it'd be easy to assume that the first release date belongs to the first album name, but how can we be sure that album names and dates are always entered in order and not changed afterward?

**Table 3.2.** Artists *and Albums: Eliminate the Repeating Group, but at What Cost?*

| **Artist Name** | **Genre** | **Album Name 1** | **Release Date 1** | **Album Name 2** | **Release Date 2** |
| --- | --- | --- | --- | --- | --- |
| The Awkward Stage | Rock | Home | 10/01/2006 | NULL | NULL |
| Girth | Metal | On the Sea | 5/25/1997 | NULL | NULL |
| Wasabi Peanuts | Adult Contemporary Rock | Spicy Legumes | 11/12/2005 | NULL | NULL |
| The Bobby Jenkins Band | R&B | Running the Game | 7/27/1985 | Live! | 10/30/1988 |
| Juices of Brazil | Latin Jazz | Long Road | 1/01/2003 | White | 6/10/2005 |

There are two ways to eliminate the problem of the repeating group. First, we could add new attributes to handle the additional albums, as in Table 3.2.

We've solved the problem of the repeating group, and because no attribute contains more than one value, this table is in 1NF. However, we've introduced a much bigger problem: what if an artist has more than two albums? Do we keep adding two attributes for each album that any artist releases? In addition to the obvious problem of adding attributes to the entity, in the physical implementation we are wasting a great deal of space for each artist who has only one album. Also, querying the resultant table for album names would require searching every album name column, something that is very inefficient.

If this is the wrong way, what's the right way? Take a look at Tables 3.3 and 3.4.

#### **Table 3.3.** *The Artists*

| **ArtistName** | **Genre** |
| --- | --- |
| The Awkward Stage | Rock |
| Girth | Metal |
| Wasabi Peanuts | Adult Contemporary Rock |
| The Bobby Jenkins Band | R&B |
| Juices of Brazil | Latin Jazz |

#### **Table 3.4.** *The Albums*

| **AlbumName** | **ReleaseDate** | **ArtistName** |
| --- | --- | --- |
| White | 6/10/2005 | Juices of Brazil |
| Home | 10/01/2006 | The Awkward Stage |
| On The Sea | 5/25/1997 | Girth |
| Spicy Legumes | 11/12/2005 | Wasabi Peanuts |
| Running the Game | 7/27/1985 | The Bobby Jenkins Band |
| Live! | 10/30/1988 | The Bobby Jenkins Band |
| Long Road | 1/01/2003 | Juices of Brazil |

We've solved the problem by adding another entity that stores album names as well the attribute that represents the relationship to the artist entity. Neither of these entities has a repeating group, each attribute in both entities holds a single value, and all of the previously mentioned query problems have been eliminated. This database is now in 1NF and ready to be deployed, right? Considering there are several other normal forms, we think you know the answer.

**Second Normal Form (2NF)**

Second normal form (2NF) specifies that, in addition to meeting 1NF, all non-key attributes have a functional dependency on the entire primary key. A functional dependency is a one-way relationship between the primary key attribute (or attributes) and all other non-key attributes in the same entity. Referring again to Table 3.3, if ArtistName is the primary key, then all other attributes in the entity must be identified by ArtistName. So we can say, "ArtistName determines ReleaseDate" for each instance in the entity. Notice that the relationship does not necessarily hold in the reverse direction; any genre may appear multiple times throughout this entity. Nonetheless, for any given artist, there is one genre. But what if an artist crosses over to another genre?

To answer that question, let's compare 1NF to 2NF. In 1NF, we have no repeating groups, and all attributes have a single value. However, in 1NF, if we have a composite primary key, it is possible that there are attributes that rely on only one of the primary key attributes, and that can lead to strange data manipulation anomalies. Take a look at Table 3.5, in which we have solved the multiple genre problem. But we have added new attributes, and that presents a new problem.

#### **Table 3.5*.*** *Artists: 1NF Is Met, but with Problems*

| **PK—Artist Name** | **PK—Genre** | **SignedDate** | **Agent** | **AgentPrimaryPhone** | **AgentSecondaryPhone** |
| --- | --- | --- | --- | --- | --- |
| The Awkward Stage | Rock | 9/01/2005 | John Doe | (777)555-1234 | NULL |
| Girth | Metal | 10/31/1997 | Sally Sixpack | (777)555-6789 | (777)555-0000 |
| Wasabi Peanuts | Adult Contemporary Rock | 1/01/2005 | John Doe | (777)555-1234 | NULL |
| The Bobby Jenkins Band | R&B | 3/15/1985 | Johnny Jenkins | (444)555-1111 | NULL |
| The Bobby Jenkins Band | Soul | 3/15/1985 | Johnny Jenkins | (444)555-1111 | NULL |
| Juices of Brazil | Latin Jazz | 6/01/2001 | Jane Doe | (777)555-4321 | (777)555-9999 |
| Juices of Brazil | World Beat | 6/01/2001 | Jane Doe | (777)555-4321 | (777)555-9999 |

In this case, we have two attributes in the primary key: Artist Name and Genre. If the studio decides to sell the Juices of Brazil albums in multiple genres to increase the band's exposure, we end up with multiple instances of the group in the entity, because one of the primary key attributes has a different value. Also, we've started storing the name of each band's agent. The problem here is that the Agent attribute is an attribute of the artist but not of the genre. So the Agent attribute is only partially dependent on the entity's primary key. If we need to update the Agent attribute for a band that has multiple entries, we must update multiple records or else risk having two different agent names listed for the same band. This practice is inefficient and risky from a data integrity standpoint. It is this type of problem that 2NF eliminates.

Tables 3.6 and 3.7 show one possible solution to our problem. In this case, we can break the entity into two different entities. The original entity still contains only information about our artists; the new entity contains information about agents and the bands they represent. This technique removes the partial dependency of the Agent attribute from the original entity, and it lets us store more information that is specific to the agent.

#### **Table 3.6.** *Artists: 2NF Version of This Entity*

| **PK—Artist Name** | **PK—Genre** | **SignedDate** |
| --- | --- | --- |
| The Awkward Stage | Rock | 9/01/2005 |
| Girth | Metal | 10/31/1997 |
| Wasabi Peanuts | Adult Contemporary Rock | 1/01/2005 |
| The Bobby Jenkins Band | R&B | 3/15/1985 |
| The Bobby Jenkins Band | Soul | 3/15/1985 |
| Juices of Brazil | Latin Jazz | 6/01/2001 |
| Juices of Brazil | World Beat | 6/01/2001 |

#### **Table 3.7**. *Agents: An Additional Entity to Solve the Problem*

| **PK—Agent Name** | **Artist Name** | **AgentPrimaryPhone** | **AgentSecondaryPhone** |
| --- | --- | --- | --- |
| John Doe | The Awkward Stage | 555-1234 | NULL |
| Sally Sixpack | Girth | (777)555-6789 | (777)555-0000 |
| Johnny Jenkins | The Bobby Jenkins Band | (444)555-1111 | NULL |
| Jane Doe | Juices of Brazil | 555-4321 | 555-9999 |

**Third Normal Form (3NF)**

Third normal form is the form that most well-designed databases meet. 3NF extends 2NF to include the elimination of transitive dependencies. Transitive dependencies are dependencies that arise from a non-key attribute relying on another non-key attribute that relies on the primary key. In other words, if there is an attribute that doesn't rely on the primary key but does rely on another attribute, then the first attribute has a transitive dependency. As with 2NF, to resolve this issue we might simply move the offending attribute to a new entity. Coincidentally, in solving the 2NF problem in Table 3.7, we also created a 3NF entity. In this particular case, AgentPrimaryPhone and AgentSecondaryPhone are not actually attributes of an artist; they are attributes of an agent. Storing them in the Artists entity created a transitive dependency, violating 3NF.

The differences between 2NF and 3NF are very subtle. 2NF deals with partial dependency, and 3NF with transitive dependency. Basically, a partial dependency means that attributes in the entity don't rely entirely on the primary key. Transitive dependency means that attributes in the entity don't rely on the primary key at all, but they do rely on another non-key attribute in the table. In either case, removing the offending attribute (and related attributes, in the 3NF case) to another entity solves the problem.

One of the simplest ways to remember the basics of 3NF is the popular phrase, "The key, the whole key, and nothing but the key." Because the normal forms are nested, the phrase means that 1NF is met because there is a primary key ("the key"), 2NF is met because all attributes in the table rely on all the attributes in the primary key ("the whole key"), and 3NF is met because none of the non-key attributes in the entity relies on any other non-key attributes ("nothing but the key"). Often, people append the phrase, "So help me Codd." Whatever helps you keep it straight.

**Boyce-Codd Normal Form (BCNF)**

In certain situations, you may discover that an entity has more than one potential, or candidate, primary key (single or composite). Boyce-Codd normal form simply adds a requirement, on top of 3NF, that states that if any entity has more than one possible primary key, then the entity should be split into multiple entities to separate the primary key attributes. For the vast majority of databases, solving the problem of 3NF actually solves this problem as well, because identifying the attribute that has a transitive dependency also tends to reveal the candidate key for the new entity being created. However, strictly speaking, the original 3NF definition did not specify this requirement, so BCNF was added to the list of normal forms to ensure that this was covered.

**Fourth Normal Form (4NF) and Fifth Normal Form (5NF)**

You've seen that 3NF generally solves most logical problems within databases. However, there are more-complicated relationships that often benefit from 4NF and 5NF. Consider Table 3.8, which describes an alternative, expanded version of the Agents entity.

**Table 3.8.** *Agents: More Agent Information*

| **PK—Agent Name** | **PK—Agency** | **PK—Artist Name** | **AgentPrimaryPhone** | **AgentSecondaryPhone** |
| --- | --- | --- | --- | --- |
| John Doe | AAA Talent | The Awkward Stage | (777)555-1234 | NULL |
| Sally Sixpack | A Star Is Born Agency | Girth | (777)555-6789 | (777)555-0000 |
| John Doe | AAA Talent | Wasabi Peanuts | (777)555-1234 | NULL |
| Johnny Jenkins | Johnny Jenkins Talent | The Bobby Jenkins Band | (444)555-1111 | NULL |
| Jane Doe | BBB Talent | Juices of Brazil | (777)555-4321 | (777)555-9999 |

Specifically, this entity stores information that creates redundancy, because there is a multivalued dependency within the primary key. A **multivalued dependency** is a relationship in which a primary key attribute, because of its relationship to another primary key attribute, creates multiple tuples within an entity. In this case, John Doe represents multiple artists. The primary key requires that the Agent Name, Agency, and Artist Name uniquely define an agent; if you don't know which agency an agent works for and if an agent quits or moves to another agency, updating this table will require multiple updates to the primary key attributes.

There's a secondary problem as well: we have no way of knowing whether the phone numbers are tied to the agent or tied to the agency. As with 2NF and 3NF, the solution here is to break Agency out into its own entity. 4NF specifies that there be no multivalued dependencies in an entity. Consider Tables 3.9 and 3.10, which show a 4NF of these entities.

#### **Table 3.9**. *Agent-Only Information*

| **PK—Agent Name** | **AgentPrimaryPhone** | **AgentSecondaryPhone** | **Artist Name** |
| --- | --- | --- | --- |
| John Doe | (777)555-1234 | NULL | The Awkward Stage |
| Sally Sixpack | (777)555-6789 | (777)555-0000 | Girth |
| John Doe | (777)555-1234 | NULL | Wasabi Peanuts |
| Johnny Jenkins | (444)555-1111 | NULL | The Bobby Jenkins Band |
| Jane Doe | (777)555-4321 | (777)555-9999 | Juices of Brazil |

**Table 3.10.** *Agency Information*

| **PK—Agency** | **AgencyPrimaryPhone** |
| --- | --- |
| AAA Talent | (777)555-1234 |
| A Star Is Born Agency | (777)555-0000 |
| AAA Talent | (777)555-4455 |
| Johnny Jenkins Talent | (444)555-1100 |
| BBB Talent | (777)555-9999 |

Now we have a pair of entities that have relevant, unique attributes that rely on their primary keys. We've also eliminated the confusion about the phone numbers.

Often, databases that are being normalized with the target of 3NF end up in 4NF, because this multivalued dependency problem is inherently obvious when you properly identify primary keys. However, the 3NF version of these entities would have worked, although it isn't necessarily the most efficient form.

Now that we have a number of 3NF and 4NF entities, we must relate these entities to one another. The final normal form that we discuss is **fifth normal form** (5NF). 5NF specifically deals with relationships among three or more entities, often referred to as **tertiary** relationships. In 5NF, the entities that have specified relationships must be able to stand alone as individual entities without dependence on the other relationships. However, because the entities relate to one another, 5NF usually requires a physical entity that acts as a resolution entity to relate the other entities to one another. This additional entity has three or more foreign keys (based on the number of entities in the relationship) that specify how the entities relate to one another. This is how many-to-many relationships (as defined in Chapter 2) are actually implemented. Thus, if a many-to-many relationship is properly implemented, the database is in 5NF.

Frequently, you can avoid the complexity of 5NF by properly implementing foreign keys in the entities that relate to one another, so 4NF plus these keys generally avoids the physical implementation of a 5NF data model. However, because this alternative is not always realistic, 5NF is defined to help formalize this scenario.

See more: <http://www.informit.com/store/product.aspx?isbn=0321497643>

# Primary and foreign keys

**Database Primary and Foreign Keys:**

**Primary and foreign keys** are the most basic components on which relational database theory is based. Primary keys enforce entity integrity by uniquely identifying entity instances. Foreign keys enforce referential integrity by completing an association between two entities. The next step in building the basic data model to:

Identify and define the primary key attributes for each entity

Validate primary keys and relationships

Migrate the primary keys to establish foreign keys

**Define Primary Key Attributes:**

Attributes are data items that describe an entity. An attribute instance is a single value of an attribute for an instance of an entity. For example, Name and hire date are attributes of the entity EMPLOYEE. "Robert Thompson" and "12 April 1999" are instances of the attributes name and hire date.

The primary key is an attribute or a set of attributes that uniquely identify a specific instance of an entity. Every entity in the data model must have a primary key whose values uniquely identify instances of the entity.

To qualify as a primary key for an entity, an attribute must have the following properties:

It must have a non-null value for each instance of the entity

The value must be unique for each instance of an entity

The values must not change or become null during the life of each entity instance

In some instances, an entity will have more than one attribute that can serve as a primary key. Any key or minimum set of keys that could be a primary key is called a candidate key. Once candidate keys are identified, choose one, and only one, primary key for each entity. Choose the identifier most commonly used by the user as long as it conforms to the properties listed above. Candidate keys which are not chosen as the primary key are known as alternate keys.

An example of an entity that could have several possible primary keys is Employee. Let's assume that for each employee in an organization there are three candidate keys: Employee ID, Social Security Number, and Name.

Name is the least desirable candidate. While it might work for a small department where it would be unlikely that two people would have exactly the same name, it would not work for a large organization that had hundreds or thousands of employees. However, there is the possibility that an employee's name could change because of marriage (your primary key should never be changed). Employee ID would be a good candidate as long as each employee were assigned a unique identifier at the time of hire. Social Security would work best since every employee is required to have one before being hired.

**Composite Keys:**

Sometimes it requires more than one attribute to uniquely identify an entity. A primary key that made up of more than one attribute is known as a composite key. Below shows an example of a composite key. Each instance of the entity Work can be uniquely identified only by a composite key composed of Employee ID and Project ID.

**Artificial Keys:**

An artificial key is one that has no meaning to the business or organization. Artificial keys are permitted when:

no attribute has all the primary key properties,

or the primary key is large and complex

|  |  |  |
| --- | --- | --- |
| EmployeeID | ProjectID | HoursWorked |
| 1 | 1 | 100 |
| 1 | 2 | 120 |
| 2 | 1 | 75 |
| 2 | 3 | 115 |
| 3 | 3 | 140 |
| 3 | 4 | 80 |

**Table 4.1** Example of Composite Keys

**Primary Key Migration:**

Dependent entities, entities that depend on the existence of another entity for their identification, inherit the entire primary key from the parent entity. Every entity within a generalization hierarchy inherits the primary key of the root generic entity.

**Define Key Attributes:**

Once the keys have been identified for the model, it is time to name and define the attributes that have been used as keys.

There is no standard method for representing primary keys in ER diagrams. For this article, the name of the primary key followed by the notation (PK) is written inside the entity box. An example is shown in below:

**Validate Keys and Relationships**

Basic rules governing the identification and migration of primary keys are:

Every entity in the data model shall have a primary key whose values uniquely identify entity instances.

The primary key attribute cannot be optional (i.e., have null values).

The primary key cannot have repeating values. That is, the attribute may not have more than one value at a time for a given entity instance is prohibited. This is known as the No Repeat Rule.

Entities with compound primary keys cannot be split into multiple entities with simpler primary keys. This is called the Smallest Key Rule.

Two entities may not have identical primary keys with the exception of entities within generalization hierarchies.

The entire primary key must migrate from parent entities to child entities and from supertype, generic entities, to subtypes, category entities.

**Foreign Keys:**

A foreign key is an attribute that completes a relationship by identifying the parent entity. Foreign keys provide a method for maintaining integrity in the data (called referential integrity) and for navigating between different instances of an entity. Every relationship in the model must be supported by a foreign key.

**Identifying Foreign Keys:**

Every dependent and category (subtype) entity in the model must have a foreign key for each relationship in which it participates. Foreign keys are formed in dependent and subtype entities by migrating the entire primary key from the parent or generic entity. If the primary key is composite, it may not be split.

**Foreign Key Ownership:**

Foreign key attributes are not considered to be owned by the entities to which they migrate, because they are reflections of attributes in the parent entities. Thus, each attribute in an entity is either owned by that entity or belongs to a foreign key in that entity.

If the primary key of a child entity contains all the attributes in a foreign key, the child entity is said to be "identifier dependent" on the parent entity, and the relationship is called an "identifying relationship." If any attributes in a foreign key do not belong to the child's primary key, the child is not identifier dependent on the parent, and the relationship is called "non identifying."

# ANSI and Oracle SQL

## SQL Standards

Oracle strives to comply with industry-accepted standards and participates actively in SQL standards committees. Industry-accepted committees are the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO), which is affiliated with the International Electrotechnical Commission (IEC). Both ANSI and the ISO/IEC have accepted SQL as the standard language for relational databases. When a new SQL standard is simultaneously published by these organizations, the names of the standards conform to conventions used by the organization, but the standards are technically identical.

The latest SQL standard was adopted in December 2011 and is often called SQL:2011. Oracle 11g Release 2 supports SQL:2008 (the previous version). The formal names of this standard are:

* ANSI/ISO/IEC 9075:2008, "Database Language SQL", Parts 1 ("SQL/Framework"), 2 ("SQL/Foundation"), 3 ("SQL/CLI"), 4 ("SQL/PSM"), 9 ("SQL/MED"), 10 ("SQL/OLB"), 11("SQL/Schemata"), 13 ("SQL/JRT"), and ANSI/ISO/IEC 9075-14:2008, "Database Language SQL", Part 14 ("SQL/XML")
* ISO/IEC 9075:2008, "Database Language SQL", Parts 1 ("SQL/Framework"), 2 ("SQL/Foundation"), 3 ("SQL/CLI"), 4 ("SQL/PSM"), 9 ("SQL/MED"), 10 ("SQL/OLB"), 11("SQL/Schemata"), 13 ("SQL/JRT"), and ISO/IEC 9075-14:2008, "Database Language SQL", Part 14 ("SQL/XML")

## Common Language for All Relational Databases

All major relational database management systems support SQL, so you can transfer all skills you have gained with SQL from one database to another. In addition, all programs written in SQL are portable. They can often be moved from one database to another with little modification.

The strengths of SQL provide benefits for all types of users, including application programmers, database administrators, managers, and end users. Technically speaking, SQL is a data sublanguage. The purpose of SQL is to provide an interface to a relational database such as Oracle Database, and all SQL statements are instructions to the database. In this SQL differs from general-purpose programming languages like C and BASIC. Among the features of SQL are the following:

* It processes sets of data as groups rather than as individual units.
* It provides automatic navigation to the data.
* It uses statements that are complex and powerful individually, and that therefore stand alone. Flow-control statements were not part of SQL originally, but they are found in the recently accepted optional part of SQL, ISO/IEC 9075-5: 1996. Flow-control statements are commonly known as "persistent stored modules" (PSM), and the PL/SQL extension to Oracle SQL is similar to PSM.

SQL lets you work with data at the logical level. You need to be concerned with the implementation details only when you want to manipulate the data. For example, to retrieve a set of rows from a table, you define a condition used to filter the rows. All rows satisfying the condition are retrieved in a single step and can be passed as a unit to the user, to another SQL statement, or to an application. You need not deal with the rows one by one, nor do you have to worry about how they are physically stored or retrieved. All SQL statements use the optimizer, a part of Oracle Database that determines the most efficient means of accessing the specified data. Oracle also provides techniques that you can use to make the optimizer perform its job better. SQL provides statements for a variety of tasks, including:

* Querying data
* Inserting, updating, and deleting rows in a table
* Creating, replacing, altering, and dropping objects
* Controlling access to the database and its objects
* Guaranteeing database consistency and integrity

SQL unifies all of the preceding tasks in one consistent language.

## ANSI Standard Structure

SQL was adopted as a standard by the American National Standards Institute (ANSI) in 1986 as SQL-86 and the International Organization for Standardization (ISO) in 1987. Nowadays the standard is subject to continuous improvement by the Joint Technical Committee ISO/IEC JTC 1, Information technology, Subcommittee SC 32, Data management and interchange which affiliate to ISO as well as IEC. It is commonly denoted by the pattern: ISO/IEC 9075-n:yyyy Part n: title, or, as a shortcut, ISO/IEC 9075.

|  |  |  |
| --- | --- | --- |
| Year | Standard Name (and Aliases) | Oracle Database |
| 1986 | SQL-86 / SQL-87 |  |
| 1989 | SQL-89 / FIPS 127-1 |  |
| 1992 | SQL-92 / SQL2 / FIPS 127-2 |  |
| 1999 | SQL:1999 / SQL3 |  |
| 2003 | SQL:2003 | Oracle 10g Release 1  Oracle 10g Release 2  Oracle 11g Release 1 |
| 2006 | SQL:2006 |  |
| **2008** | **SQL:2008** | **Oracle 11g Release 2** |
| 2011 | SQL:2011 |  |

Table 1: SQL Standards

The SQL standard is divided into nine parts.

**ISO/IEC 9075-1:2011 Part 1: Framework (SQL/Framework).** It provides logical concepts.

**ISO/IEC 9075-2:2011 Part 2: Foundation (SQL/Foundation).** It contains the most central elements of the language and consists of both mandatory and optional features.

**ISO/IEC 9075-3:2008 Part 3: Call-Level Interface (SQL/CLI).** It defines interfacing components (structures, procedures, variable bindings) that can be used to execute SQL statements from applications written in Ada, C respectively C++, COBOL, Fortran, MUMPS, Pascal or PL/I. (For Java see part 10.) SQL/CLI is defined in such a way that SQL statements and SQL/CLI procedure calls are treated as separate from the calling application's source code. Open Database Connectivity is a well-known superset of SQL/CLI. This part of the standard consists solely of mandatory features.

**ISO/IEC 9075-4:2011 Part 4: Persistent Stored Modules (SQL/PSM).** It standardizes procedural extensions for SQL, including flow of control, condition handling, statement condition signals and resignals, cursors and local variables, and assignment of expressions to variables and parameters. In addition, SQL/PSM formalizes declaration and maintenance of persistent database language routines (e.g., "stored procedures"). This part of the standard consists solely of optional features.

**ISO/IEC 9075-9:2008 Part 9: Management of External Data (SQL/MED).** It provides extensions to SQL that define foreign-data wrappers and datalink types to allow SQL to manage external data. External data is data that is accessible to, but not managed by, an SQL-based DBMS. This part of the standard consists solely of optional features.

**ISO/IEC 9075-10:2008 Part 10: Object Language Bindings (SQL/OLB).** It defines the syntax and semantics of SQLJ, which is SQL embedded in Java (see also part 3). The standard also describes mechanisms to ensure binary portability of SQLJ applications, and specifies various Java packages and their contained classes. This part of the standard consists solely of optional features. As opposed to SQL/OLB JDBC - which is not part of the SQL standard - defines an API.

**ISO/IEC 9075-11:2011 Part 11: Information and Definition Schemas (SQL/Schemata).** It defines the Information Schema and Definition Schema, providing a common set of tools to make SQL databases and objects self-describing. These tools include the SQL object identifier, structure and integrity constraints, security and authorization specifications, features and packages of ISO/IEC 9075, support of features provided by SQL-based DBMS implementations, SQL-based DBMS implementation information and sizing items, and the values supported by the DBMS implementations. This part of the standard contains both mandatory and optional features.

**ISO/IEC 9075-13:2008 Part 13: SQL Routines and Types Using the Java Programming Language (SQL/JRT).** It specifies the ability to invoke static Java methods as routines from within SQL applications ('Java-in-the-database'). It also calls for the ability to use Java classes as SQL structured user-defined types. This part of the standard consists solely of optional features.

**ISO/IEC 9075-14:2011 Part 14: XML-Related Specifications (SQL/XML).** It specifies SQL-based extensions for using XML in conjunction with SQL. The XMLType data type is introduced, as well as several routines, functions, and XML-to-SQL data type mappings to support manipulation and storage of XML in an SQL database. This part of the standard consists solely of optional features.

## Oracle Compliance To Core SQL:2008

The ANSI and ISO SQL standards require conformance claims to state the type of conformance and the implemented facilities. The minimum claim of conformance is called **Core SQL:2008** and is defined in Part 2, SQL/Foundation, and Part 11, SQL/Schemata, of the standard.

The SQL standards conformance features can be used either as a guide to portability, or as a guide to functionality. From the standpoint of portability, the user is interested in conformance to both the precise syntax and semantics of the standard feature. From the standpoint of functionality, the user is less concerned about the precise syntax and more concerned with issues of semantics.

When we will talk about ANSI SQL and Oracle SQL conformance we will use the following terms regarding support for standard syntax and semantics:

* **Full Support**: The feature is supported with standard syntax and semantics.
* **Partial Support**: Some, but not all, of the standard syntax is supported; whatever is supported has standard semantics.
* **Enhanced Support**: The standard semantics is supported, but gives functionality that differs from the standard by enhancing it.
* **Equivalent Support**: The standard semantics is supported using non-standard syntax.
* **Similar Support**: Neither the standard's syntax nor semantics are supported precisely, but similar functionality is provided.

## Character Sets Support

Oracle supports most national, international, and vendor-specific encoded character set standards. A complete list of character sets supported by Oracle appears in Oracle Database Globalization Support Guide. Unicode is a universal encoded character set that lets you store information from any language using a single character set. Unicode is strongly recommended or required by modern standards such as XML, Java, JavaScript, and LDAP. Unicode is compliant with ISO/IEC standard 10646.

Oracle Database complies fully with Unicode 4.0, the fourth and most recent version of the Unicode standard (see <http://www.unicode.org>). Oracle uses UTF-8 (8-bit) encoding by way of three database character sets, two for ASCII-based platforms (UTF8 and AL32UTF8) and one for EBCDIC platforms (UTFE). If you prefer to implement Unicode support incrementally, then you can store Unicode data in either the UTF-16 or UTF-8 encoding form, in the national character set, for the SQL NCHAR data types (NCHAR, NVARCHAR2, and NCLOB).



# Oracle SQL

## Basic Language Elements

* **Clauses** are constituent components of statements and queries.
* **Expressions**, which can produce either scalar values or tables consisting of columns and rows of data.
* **Predicates**, which specify conditions that can be evaluated to SQL three-valued logic (3VL) or Boolean (true/false/unknown) truth values and which are used to limit the effects of statements and queries, or to change program flow.
* **Queries**, which retrieve the data based on specific criteria. This is the most important element of SQL.
* **Statements**, which may have a persistent effect on schemata and data, or which may control transactions, program flow, connections, sessions, or diagnostics. SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.
* **Insignificant whitespace** is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

## Oracle Database Objects Naming Rules (simplified)

The names of Oracle identifiers, such as tables and columns, must not exceed 30 characters in length. The first character must be a letter, but the rest can be any combination of letters, numerals, dollar signs ($), pound signs (#), and underscores (\_).

However, if an Oracle identifier is enclosed by double quotation marks ("), then it can contain any combination of legal characters, including spaces but excluding quotation marks. Oracle identifiers are not case sensitive except within double quotation marks.

See [1]: "Database Object Naming Rules" on page 3-109 for more information.

**Oracle Identifiers and ANSI**

Oracle supports ANSI “E031, Identifiers” feature, with the following exceptions:

* Oracle does not support the escape sequence to permit a double quote within a quoted identifier
* A non-quoted identifier may not be equivalent to an Oracle reserved word (the list of Oracle reserved words differs from the standard's list)
* A column name may not be ROWID, even as a quoted identifier

Oracle extends this feature as follows:

* An identifier may be up to 30 characters long
* A non-quoted identifier may have dollar sign ($) or pound sign (#)

## Oracle support for SELECT statement clauses

Oracle fully supports the following subfeatures:

* E051-01, SELECT DISTINCT
* E051-02, GROUP BY clause
* E051-04, GROUP BYcan contain columns not in SELECT list
* E051-05, SELECT list items can be renamed
* E051-06, HAVING clause
* E051-07, Qualified \* in SELECT list

Oracle partially supports the following subfeatures:

* E051-08, Correlation names in FROM clause (Oracle supports correlation names, but not the optional AS keyword)

Oracle has equivalent functionality for the following subfeature:

* E051-09, Rename columns in the FROM clause (column names can be renamed in a subquery in the FROM clause)

## Dummy table (DUAL)

By ANSI standard FROM clause is required in query construction:

<query specification> ::= SELECT[ <set quantifier>] <select list> <table expression>

<table expression> ::= <from clause> [ <where clause>] [ <group by clause>] [ <window clause>]

Oracle supports this standard and provide special dummy table DUAL.

DUAL is a table automatically created by Oracle Database along with the data dictionary. DUAL is in the schema of the user SYS but is accessible by the name DUAL to all users. It has one column, DUMMY, defined to be VARCHAR2(1), and contains one row with a value X. Selecting from the DUAL table is useful for computing a constant expression with the SELECT statement. Because DUAL has only one row, the constant is returned only once. Alternatively, you can select a constant, pseudo column, or expression from any table, but the value will be returned as many times as there are rows in the table.

## Set Operations

ANSI Standard declares next set operations:

* UNION[ ALL| DISTINCT]
* EXCEPT[ ALL| DISTINCT]
* INTERSECT[ ALL| DISTINCT]

Oracle fully supports the following subfeatures of ANSI “E071, Basic query expressions”:

* E071-01, UNION DISTINCT table operator
* E071-02, UNION ALL table operator
* E071-05, Columns combined by table operators need not have exactly the same type
* E071-06, table operators in subqueries

Oracle has equivalent functionality for the following subfeature:

* E071-03, EXCEPT DISTINCT table operator: Use MINUS instead of EXCEPT DISTINCT

|  |  |  |
| --- | --- | --- |
| **Set Operation** | **ANSI Standard** | **Oracle** |
| UNION | UNION ALL | UNION ALL |
|  | UNION DISTINCT | UNION |
| INTERSECT | INTERSECT ALL |  |
|  | INTERSECT DISTINCT | INTERSECT |
| EXCEPT | EXCEPT ALL |  |
|  | EXCEPT DISTINCT | MINUS |

Table 2: Set Operations

# Pseudocolumns

A pseudocolumnbehaves like a table column, but is not actually stored in the table. You can select from pseudocolumns, but you cannot insert, update, or delete their values. A pseudocolumn is also similar to a function without arguments. However, functions without arguments typically return the same value for every row in the result set, whereas pseudocolumns typically return a different value for each row.

Oracle uses several types of pseudocolumns to perform specific actions:

* Hierarchical Query Pseudocolumns
* Sequence Pseudocolumns
* Version Query Pseudocolumns
* COLUMN\_VALUE Pseudocolumn
* OBJECT\_ID Pseudocolumn
* OBJECT\_VALUE Pseudocolumn
* ORA\_ROWSCN Pseudocolumn
* ROWID Pseudocolumn
* ROWNUM Pseudocolumn
* XMLDATA Pseudocolumn

## ROWNUM Pseudocolumn

For each row returned by a query, the ROWNUM pseudocolumn returns a number indicating the order in which Oracle selects the row from a table or set of joined rows. The first row selected has a ROWNUM of 1, the second has 2, and so on. You can use ROWNUM to limit the number of rows returned by a query, as in this example:

SELECT \*

FROM employees

WHERE ROWNUM < 11;

If an ORDER BY clause follows ROWNUM in the same query, then the rows will be reordered by the ORDER BY clause. The results can vary depending on the way the rows are accessed. Therefore, the following statement does not necessarily return the same rows as the preceding example:

SELECT \*

FROM employees

WHERE ROWNUM < 11

ORDER BY last\_name;

If you embed the ORDER BY clause in a subquery and place the ROWNUM condition in the top-level query, then you can force the ROWNUM condition to be applied after the ordering of the rows. For example, the following query returns the employees with the 10 smallest salaries. This is sometimes referred to as top-N reporting:

SELECT \*

FROM (SELECT \* FROM employees ORDER BY salary)

WHERE ROWNUM < 11;

In the preceding example, the ROWNUM values are those of the top-level SELECT statement, so they are generated after the rows have already been ordered by salary in the subquery.

*Conditions testing for ROWNUM values greater than a positive integer are always false!*

For example, this query returns no rows:

SELECT \*

FROM employees

WHERE ROWNUM > 1;

The first row fetched is assigned a ROWNUM of 1 and makes the condition false. The second row to be fetched is now the first row and is also assigned a ROWNUM of 1 and makes the condition false. All rows subsequently fail to satisfy the condition, so no rows are returned.

You can also use ROWNUM to assign unique values to each row of a table, as in this example:

UPDATE my\_table

SET column1 = ROWNUM;

## ROWID Pseudocolumn

For each row in the database, the ROWID pseudocolumn returns the address of the row. Oracle Database rowid values contain information necessary to locate a row:

* The data object number of the object
* The data block in the data file in which the row resides
* The position of the row in the data block (first row is 0)
* The data file in which the row resides (first file is 1). The file number is relative to the tablespace.

Usually, a rowid value uniquely identifies a row in the database. However, rows in different tables that are stored together in the same cluster can have the same rowid.

Rowid values have several important uses:

* *They are the fastest way to access a single row.*
* They can show you how the rows in a table are stored.
* They are unique identifiers for rows in a table.

*You should not use ROWID as the primary key of a table.* If you delete and reinsert a row with the Import and Export utilities, for example, then its rowid may change. If you delete a row, then Oracle may reassign its rowid to a new row inserted later. Although you can use the ROWID pseudocolumn in the SELECT and WHERE clause of a query, these pseudocolumn values are not actually stored in the database. You cannot insert, update, or delete a value of the ROWID pseudocolumn.

For manipulating with ROWIDs there is special package DBMS\_ROWID. See “Using DBMS\_ROWID” in [2].

# Client Tools Support

## Client Tools Overview

Oracle provides a number of utilities to facilitate your SQL development process:

* Oracle SQL Developer is a graphical tool that lets you browse, create, edit, and delete (drop) database objects, edit and debug PL/SQL code, run SQL statements and scripts, manipulate and export data, and create and view reports. With SQL Developer, you can connect to any target Oracle Database schema using standard Oracle Database authentication. Once connected, you can perform operations on objects in the database. You can also connect to schemas for selected third-party (non-Oracle) databases, such as MySQL, Microsoft SQL Server, and Microsoft Access, view metadata and data in these databases, and migrate these databases to Oracle.
* SQL\*Plus is an interactive and batch query tool that is installed with every Oracle Database server or client installation. It has a command-line user interface and a Web-based user interface called iSQL\*Plus.
* Oracle JDeveloper is a multiple-platform integrated development environment supporting the complete lifecycle of development for Java, Web services, and SQL. It provides a graphical interface for executing and tuning SQL statements and a visual schema diagrammer (database modeler). It also supports editing, compiling, and debugging PL/SQL applications.
* Oracle Application Express is a hosted environment for developing and deploying database-related Web applications. SQL Workshop is a component of Oracle Application Express that lets you view and manage database objects from a Web browser. SQL Workshop offers quick access to a SQL command processor and a SQL script repository.
* The Oracle Call Interface and Oracle precompilers let you embed standard SQL statements within a procedure programming language. The Oracle Call Interface (OCI) lets you embed SQL statements in C programs. The Oracle precompilers, Pro\*C/C++ and Pro\*COBOL, interpret embedded SQL statements and translate them into statements that can be understood by C/C++ and COBOL compilers, respectively.
* Oracle provides ODBC and JDBC drivers to access Oracle database through ODBC or JDBC protocols respectively.
* Oracle Data Provider for .NET (ODP.NET) features optimized ADO.NET data access to the Oracle database. ODP.NET allows developers to take advantage of advanced Oracle database functionality, including Real Application Clusters, XML DB, and advanced security. The data provider can be used with the latest .NET Framework 4 version.

## Oracle SQL Developer

### Download and Install

Download the latest version of Oracle SQL Developer from <http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html>

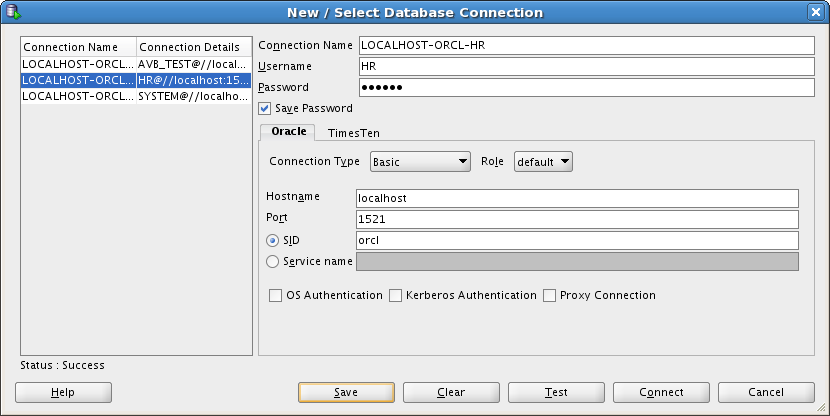
The easiest way is to get binary named “[Oracle SQL Developer for 32-bit Windows](http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html) (This zip file includes the JDK1.6.0\_11)”.

Oracle Learning Library resources:

* [Getting Started with Oracle SQL Developer 3.0](http://apex.oracle.com/pls/apex/f?p=44785:24:896077371487201::NO:24:P24_CONTENT_ID,P24_PREV_PAGE:5234,2)
* [Working with Data Grids using Oracle SQL Developer 3.0](http://apex.oracle.com/pls/apex/f?p=44785:24:896077371487201::NO:24:P24_CONTENT_ID,P24_PREV_PAGE:5249,2)
* [Using Query Builder in Oracle SQL Developer 3.0](http://apex.oracle.com/pls/apex/f?p=44785:24:896077371487201::NO:24:P24_CONTENT_ID,P24_PREV_PAGE:5235,2)

### Establish Connection

It’s strongly recommended to test connection before saving it.



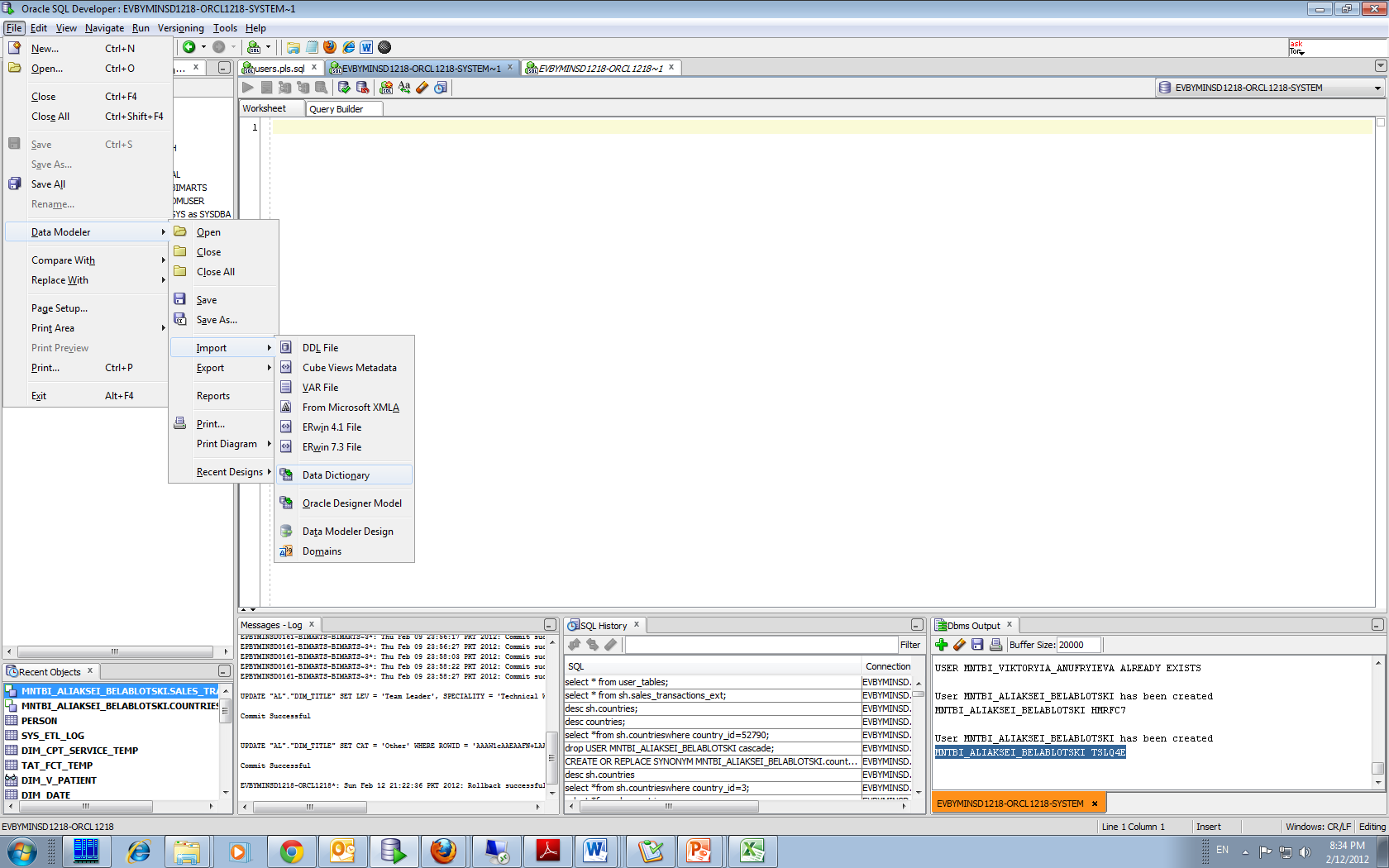
The “Connection name” can be any user-friendly string. I use SERVER-SERVICE-USER format which is convenient for me.

### Oracle Data Modeler

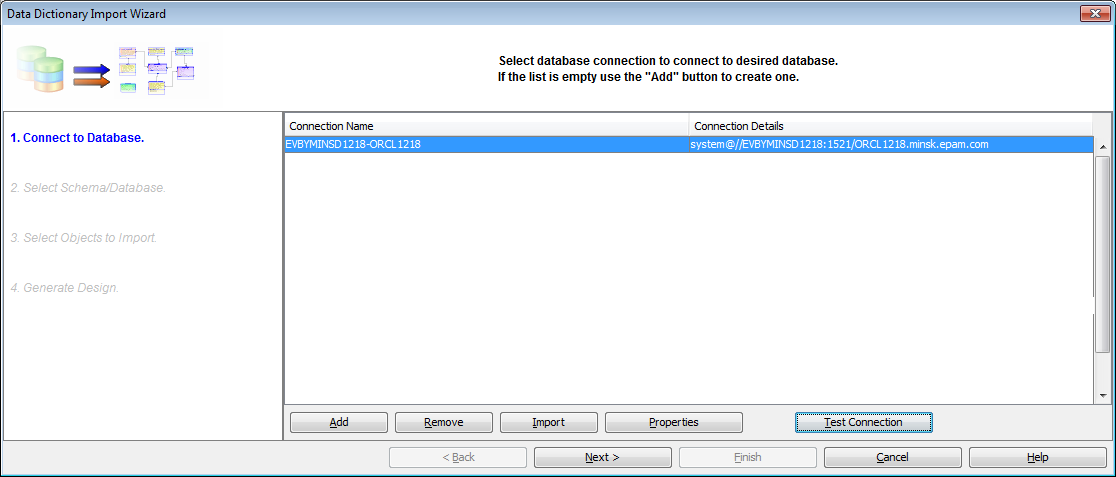
Oracle Learning Library resources:

* [SQL Developer Data Modeler 3.0 New Features Overview](http://apex.oracle.com/pls/apex/f?p=44785:24:896077371487201::NO:24:P24_CONTENT_ID,P24_PREV_PAGE:5221,2)
* [Importing Data Models](http://apex.oracle.com/pls/apex/f?p=44785:24:896077371487201::NO:24:P24_CONTENT_ID,P24_PREV_PAGE:5220,2)
* [Forward and Reverse Engineering Models and Working with Logical Model Diagrams, Displays and Subviews](http://apex.oracle.com/pls/apex/f?p=44785:24:896077371487201::NO:24:P24_CONTENT_ID,P24_PREV_PAGE:5231,2)
* [Generating a Multi-Dimensional Model with SQL Developer Data Modeler 3.0](http://apex.oracle.com/pls/apex/f?p=44785:24:896077371487201::NO:24:P24_CONTENT_ID,P24_PREV_PAGE:5229,2)
* [Adding a Table to An Existing Database for SQL Developer Data Modeler 3.0](http://apex.oracle.com/pls/apex/f?p=44785:24:896077371487201::NO:24:P24_CONTENT_ID,P24_PREV_PAGE:5228,2)

Start Data Modeler with importing metadata from existing schema (import metadata from Human Resources (HR) schema).



Use an existing connection to Human Resources (HR) schema.



Explore database objects and relationships between them. Notice “Comments in RDBMS” tab in Properties window (these comments are very helpful for understanding object purpose).

# External Sources

1. Oracle Database SQL Language Reference 11g Release 2 (11.2) E26088-02
2. Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2) B28419-03
3. ISO/IEC 9075-1 Part 1 Framework (SQL/Framework)
4. ISO/IEC 9075-2 Part 2 Foundation (SQL/Foundation)
5. ISO/IEC 9075-11 Part 11 Information and Definition Schemas (SQL/Schemata)
6. **Oracle Database SQL Language Reference 12g Release12 (12.1) E17209-15**