



Application Environment: Introduction

New Requirements for Database Applications

- A database system that has the scalability and robustness of current relational database systems is an essential component of a large-scale enterprise computing environment.
- In addition, the database must be able to easily store complex, structured data and large, unstructured, domain-specific data, such as text, image, audio and video.
- It also needs to provide query capabilities for all types of data.
- At the same time, given that current users of database technology have made huge investments in building relational applications, this needs to be accomplished in such a manner that existing relational applications, schemas and data can co-exist with new object-based schemas, data and applications.
- Current relational database technology is able to meet enterprise requirements for scalability, replication and data distribution.

Design Goals of the Oracle

- Provide users with the ability to model their business objects in the database by enhancing the type system to provide support for user-defined types.
- These types are meant to closely model application objects and are treated as built-in types, such as number and character, by the database server.
- Provide an infrastructure to facilitate object-based access to object data stored in an Oracle database.
- Provide built-in support for new data types needed in multi-media, financial and spatial applications.

User-Defined Data Types

- Oracle provides the capability to define new types in the database via a new meta-model, simply called the Oracle8 Type System.
- This type system extends the Oracle relational types to allow users to define types and methods that model objects.
- An object type is a single, basic representation mechanism that captures both the structure and behavior of a user-defined type.

Inheritance

- A major benefit of object orientation is inheritance, i.e., the ability to extend the specification and behavior of an object.
- Oracle type system will support inheritance on types.
- A type can be extended by creating subtypes which would reuse the specification and implementation (attributes and methods) of the type it derives from (parent).
- In a subtype, the ability to enhance the structure of the data as inherited from its supertype by defining other attributes is referred to as data inheritance.
- The ability to add other methods in addition to ones inherited from its supertype is referred to as method inheritance

Collection Types

- Oracle8's type system supports varying length arrays and nested tables (multisets of rows) as collection types.
- Attributes of types and columns of tables can be of a collection type.
- By using varying arrays and nested tables, application developers can model one-to-many and many-to-many relationships natively in their database schema.

Object View of Relational Data

- In addition to natively storing object data in the server, Oracle allows the creation of an object abstraction over existing relational data via the object view mechanism.
- Objects belonging to an object view are accessed in the same manner as row objects via SQL or other call interfaces

Server Programmability

- Oracle8 allows methods to be implemented in PL/SQL, C/C++ or Java.
- De-coupling of the specification of a method in SQL from its implementation provides a uniform way to invoke methods on object types, even though these object types can be implemented in various programming languages.

External Procedures

- If any function, procedure, or member method of an object type is implemented in C, then it is called an external procedure.
- External procedures are best suited for tasks that are more quickly or easily done in a low-level language such as C, which is more efficient at machine-precision calculation.

Java as a Stored Procedure Language

- Oracle is developing a high-performance, language-neutral virtual machine (VM) and an object runtime that supports Java and will eventually integrate Java, SQL, and PL/SQL in the database server.
- This runtime is compatible with the RDBMS, any Java-enabled application server and client environments.

Object Cache

- Oracle8 provides an object cache for efficient access to persistent objects stored in the database.
- Copies of objects can be brought into the object cache. Once the data has been cached in the client, the application can traverse through these at memory speed.
- Any changes made to objects in the cache can be committed to the database by using the Oracle extensions to the programmatic interfaces.
- The object cache gives applications the following additional functionality:
 - Transparent mapping of database objects to host language objects in memory.
 - Transparent, efficient memory management for persistent objects. Applications do not have to worry about allocation of memory for accessing database objects.

Relational Data Access from Java

- Oracle provides two sets of application programmatic interfaces (APIs) by which Java programmers may conveniently and efficiently access relational data: JDBC and JSQL.
- JDBC is a simple call-level interface which cannot exploit RDBMS schema information at compile time; it is a relatively low-level and sometimes tedious interface.
- JSQL supports embedded SQL for Java. It complements the capabilities of JDBC by providing Java with compile-time access to the RDBMS schema. This greatly simplifies application code, provides type safety and higher performance for data access.

Designing Tables Guidelines

- Use descriptive names for tables, columns, indexes, and clusters.
- Be consistent in abbreviations and in the use of singular and plural forms of table names and columns.
- Document the meaning of each table and its columns with the COMMENT command.
- Normalize each table.
- Select the appropriate datatype for each column.
- Define columns that allow nulls last, to conserve storage space.
- Cluster tables whenever appropriate, to conserve storage space and optimize performance of SQL statements.
- Before creating a table, you should also determine whether to use integrity constraints. Integrity constraints can be defined on the columns of a table to enforce the business rules of your database automatically.

Integrity Constraints

- You can define integrity constraints to enforce business rules on data in your tables.
- Business rules specify conditions and relationships that must always be true, or must always be false.
- Because each company defines its own policies about things like salaries, employee numbers, inventory tracking, and so on, you can specify a different set of rules for each database table.
- When an integrity constraint applies to a table, all data in the table must conform to the corresponding rule.
- When you issue a SQL statement that modifies data in the table, Oracle ensures that the new data satisfies the integrity constraint, without the need to do any checking within your program.

Creating Indexes for Use with Constraints

- All enabled unique and primary keys require corresponding indexes. You should create these indexes by hand, rather than letting the database create them for you.
- Constraints use existing indexes where possible, rather than creating new ones.
- Unique and primary keys can use non-unique as well as unique indexes. They can even use just the first few columns of non-unique indexes.
- At most one unique or primary key can use each non-unique index.
- The column orders in the index and the constraint do not need to match.

When to Use NOT NULL Integrity Constraints

- By default, all columns can contain nulls.
- Define NOT NULL constraints for columns of a table that absolutely require values at all times.
- NOT NULL constraints are often combined with other types of integrity constraints to further restrict the values that can exist in specific columns of a table.
 - Use the combination of NOT NULL and UNIQUE key integrity constraints to force the input of values in the UNIQUE key; this combination of data integrity rules eliminates the possibility that any new row's data will ever attempt to conflict with an existing row's data.

When to Use Default Column Values

- Assign default values to columns that contain a typical value.
- For example, in the DEPT_TAB table, if most departments are located at one site, then the default value for the LOC column can be set to this value (such as NEW YORK).
- Default values can help avoid errors where there is a number, such as zero, that applies to a column that has no entry.

Choosing a Table's Primary Key

- Each table can have one primary key, which uniquely identifies each row in a table and ensures that no duplicate rows exist.
- Whenever practical, use a column containing a sequence number. It is a simple way to satisfy all the other guidelines.
- Minimize your use of composite primary keys.
- Choose a column whose data values are unique, because the purpose of a primary key is to uniquely identify each row of the table.
- Choose a column whose data values are never changed. A primary key value is only used to identify a row in the table, and its data should never be used for any other purpose. Therefore, primary key values should rarely or never be changed.
- Choose a column that does not contain any nulls.

List Table and Constraints

```
-- List tables and constraints
SELECT Constraint_name,
       Constraint_type, Table_name,
       R_constraint_name
FROM User_constraints;
```

```
SELECT Constraint_name, Table_name,
       Column_name
FROM User_cons_columns;
```

Guidelines for Application-Specific Indexes

- Indexes are used in Oracle to provide quick access to rows in a table.
- Indexes provide faster access to data for operations that return a small portion of a table's rows.
- Although Oracle allows an unlimited number of indexes on a table, the indexes only help if they are used to speed up queries. Otherwise, they just take up space and add overhead when the indexed columns are updated.
- You should use the EXPLAIN PLAN feature to determine how the indexes are being used in your queries.
- Sometimes, if an index is not being used by default, you can use a query hint so that the index is used.

Create Indexes After Inserting Table Data

- Typically, you insert or load data into a table (using SQL*Loader or Import) before creating indexes.
- Otherwise, the overhead of updating the index slows down the insert or load operation.
- The exception to this rule is that you must create an index for a cluster before you insert any data into the cluster.

Index the Correct Tables and Columns

- Create an index if you frequently want to retrieve less than 15% of the rows in a large table. The percentage varies greatly according to the relative speed of a table scan and how clustered the row data is about the index key.
- The faster the table scan, the lower the percentage; the more clustered the row data, the higher the percentage.
- Index columns used for joins to improve performance on joins of multiple tables.
- Primary and unique keys automatically have indexes, but you might want to create an index on a foreign key.
- Small tables do not require indexes; if a query is taking too long, then the table might have grown from small to large.

Index the Correct Tables and Columns

- Some columns are strong candidates for indexing.
- Columns with one or more of the following characteristics are candidates for indexing:
 - Values are relatively unique in the column.
 - There is a wide range of values (good for regular indexes).
 - There is a small range of values (good for bitmap indexes).

Choose the Order of Columns in Composite Indexes

- Although you can specify columns in any order in the `CREATE INDEX` command, the order of columns in the `CREATE INDEX` statement can affect query performance.
- In general, you should put the column expected to be used most often first in the index.
- You can create a composite index (using several columns), and the same index can be used for queries that reference all of these columns, or just some of them.

Gather Statistics to Make Index Usage More Accurate

- The database can use indexes more effectively when it has statistical information about the tables involved in the queries.
- You can gather statistics when the indexes are created by including the keywords `COMPUTE STATISTICS` in the `CREATE INDEX` statement.
- As data is updated and the distribution of values changes, you or the DBA can periodically refresh the statistics by calling procedures like `DBMS_STATS.GATHER_TABLE_STATISTICS` and `DBMS_STATS.GATHER_SCHEMA_STATISTICS`.

Drop Indexes That Are No Longer Required

- You might drop an index if:
 - It does not speed up queries. The table might be very small, or there might be many rows in the table but very few index entries.
 - The queries in your applications do not use the index.
 - The index must be dropped before being rebuilt.
 - If you drop a table, then all associated indexes are dropped.

Designing Triggers

- Use the following guidelines when designing your triggers:
 - Use triggers to guarantee that when a specific operation is performed, related actions are performed.
 - Do not define triggers that duplicate features already built into Oracle. For example, do not define triggers to reject bad data if you can do the same checking through declarative integrity constraints.
 - Limit the size of triggers. If the logic for your trigger requires much more than 60 lines of PL/SQL code, it is better to include most of the code in a stored procedure and call the procedure from the trigger.
 - Use triggers on DATABASE judiciously. They are executed for *every* user *every* time the event occurs on which the trigger is created.

Simplicity in Application Design

- Well-designed structures, machines, and tools are usually reliable, easy to use and maintain, and simple in concept. In the most general terms, if the design looks right, then it probably is right.
- If the table design is so complicated that nobody can fully understand it, then the table is probably designed badly.
- If SQL statements are so long and involved that it would be impossible for any optimizer to effectively optimize it in real time, then there is probably a bad statement, underlying transaction, or table design.
- If there are indexes on a table and the same columns are repeatedly indexed, then there is probably a bad index design.
- If the calls to the database are abstracted away from the application logic by many layers of software, then there is probably a bad software development method.

Using a Different Index Type

- **B-Tree Indexes**
 - These are the standard index type, and they are excellent for primary key and highly-selective indexes. Used as concatenated indexes, B-tree indexes can be used to retrieve data sorted by the index columns.
- **Bitmap Indexes**
 - These are suitable for low cardinality data. Through compression techniques, they can generate a large number of rowids with minimal I/O. Combining bitmap indexes on non-selective columns allows efficient AND and OR operations with a great number of rowids with minimal I/O. Bitmap indexes are particularly efficient in queries with COUNT(), because the query can be satisfied within the index.
- **Function-based Indexes**
 - These indexes allow access through a B-tree on a value derived from a function on the base data. Function-based indexes have some limitations with regards to the use of nulls, and they require that you have the cost-based optimizer enabled.
 - Function-based indexes are particularly useful when querying on composite columns to produce a derived result or to overcome limitations in the way data is stored in the database. An example of this is querying for line items in an order exceeding a certain value derived from (sales price - discount) x quantity, where these were columns in the table. Another example is to apply the UPPER function to the data to allow case insensitive searches.
- **Partitioned Indexes**
 - Partitioning a global index allows partition pruning to take place within an index access, which results in reduced I/Os. By definition of good range or list partitioning, fast index scans of the correct index partitions can result in very fast query times.
- **Reverse Key Indexes**
 - These are designed to eliminate index hot spots on insert applications. These indexes are excellent for insert performance, but they are limited in that they cannot be used for index range scans.

Configuration vs Tuning

- **What can be configured:**
 - O/S
 - Physical storage
 - Database objects:
 - Tables
 - Indexes
 - Sort areas
 - Temp tablespaces
 - Redo logs
 - Database buffers
 - Redo log buffers
 - Shared pool
- **What can be tuned:**
 - Memory utilization
 - Disk utilization
 - SQL statement execution

Oracle Manuals

<http://otn.oracle.com/pls/db92/db92.homepage>

<http://otn.oracle.com/documentation/oracle9i.html>

Oracle Books



Oracle9i: The Complete Reference
by Kevin Loney, George Koch
Publisher: McGraw-Hill Osborne Media; Book and CD edition
(August 16, 2002)
ISBN: 0072225211

Oracle PL/SQL Programming, Third Edition
by Steven Feuerstein
Publisher: O'Reilly & Associates; 3rd edition (September 2002)
ISBN: 0596003811

Oracle9i DBA Handbook
by Kevin Loney, Marlene Theriault
Publisher: McGraw-Hill Osborne Media; 1st edition (November 28,
2001)
ISBN: 0072193743

Oracle9i Performance Tuning Tips & Techniques
by Richard J. Niemiec
Publisher: McGraw-Hill Osborne Media; (May 12, 2003)
ISBN: 0072224738

Oracle9i High-Performance Tuning with STATSPACK
by Donald K. Burleson, Don Burleson
Publisher: McGraw-Hill Osborne Media; (March 22, 2002)
ISBN: 007222360X

Oracle9i PL/SQL Programming
by Scott Urman
Publisher: McGraw-Hill Osborne Media; 2nd edition (November 28,
2001)
ISBN: 0072191473

Oracle SQL*Plus: The Definitive Guide
by Jonathan Gennick
Publisher: O'Reilly & Associates; (March 1999)
ISBN: 1565925785

Oracle SQL High-Performance Tuning (2nd Edition)
by Guy Harrison
Publisher: Prentice Hall PTR; 2nd edition (December 29, 2000)
ISBN: 0130123811