CERN

Database and application design

Katarzyna Dziedziniewicz-Wojcik
IT-DB

Agenda

- Database design
- PL/SQL tips and tricks
- Robust application design

Agenda

- **Database design**
  - **Schema design**
  - Integrity constraints
  - Best practices
- PL/SQL tips and tricks
- Robust application design

“It’s a Database, not a Data Dump”

- Database is **an integrated collection of logically related data**
- You need a database to:
  - Store data…
  - … and be able to efficiently process it in order to retrieve/produce information!

Design goals

- Store data and…
  - Avoid unnecessary redundancy
    - Storage is not unlimited
    - Redundant data is not logically related
  - Retrieve information easily and efficiently
    - Easily – does not necessarily mean with a simple query
    - Efficiently – using built-in database features
  - Be scalable for data and interfaces
    - **Performance** is in the **design**!
    - Will your design scale to predicted workload (thousands of connections)?

Conceptual design

- Process of constructing a model of the information used in an enterprise
- Is a conceptual representation of the data structures
- Is independent of all physical considerations

- *Input:* database requirements
- *Output:* conceptual model

Conceptual design in practice

- The Entity-Relationship model (ER) is most common conceptual model for database design:
  - Describes the data in a system and how data is related
  - Describes data as entities, attributes, and relationships
  - Can be easily translated into many database implementations
    - Oracle SQL Developer Data Modeler does it for free

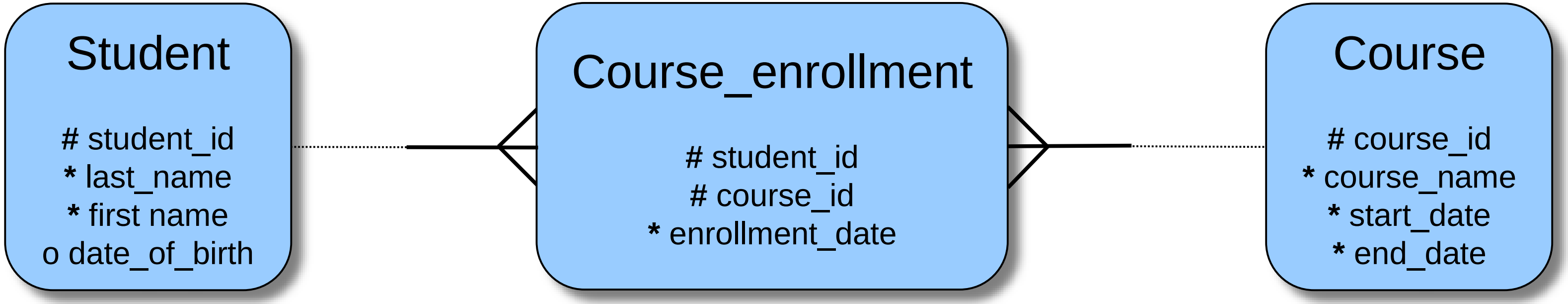
Sales
TimeKey
CustomerKey
ShipperKey
ProductKey
EmployeeKey
ItemFreight
ItemQuanity
ItemAll
Discount
Shipper
ShipperKey
ShipperID
ShipperName
Product
ProductKey
ProductID
Computed
Region
PCode
Country
Phone
Mphone
Fax
Email

Let’s get real

- Assume you have to design a database for a university/college and want to handle enrollments
- You have the courses taught, each course has a title and a regular timeslot each week
- Each course has many students who study the course
- Each student attends many courses

Modeling relationships - example

- Many – to – many (M:N)
  - A student can be registered on any number of courses (including zero)
  - A course can be taken by any number of students (including zero)
- Logical model – normalized form:

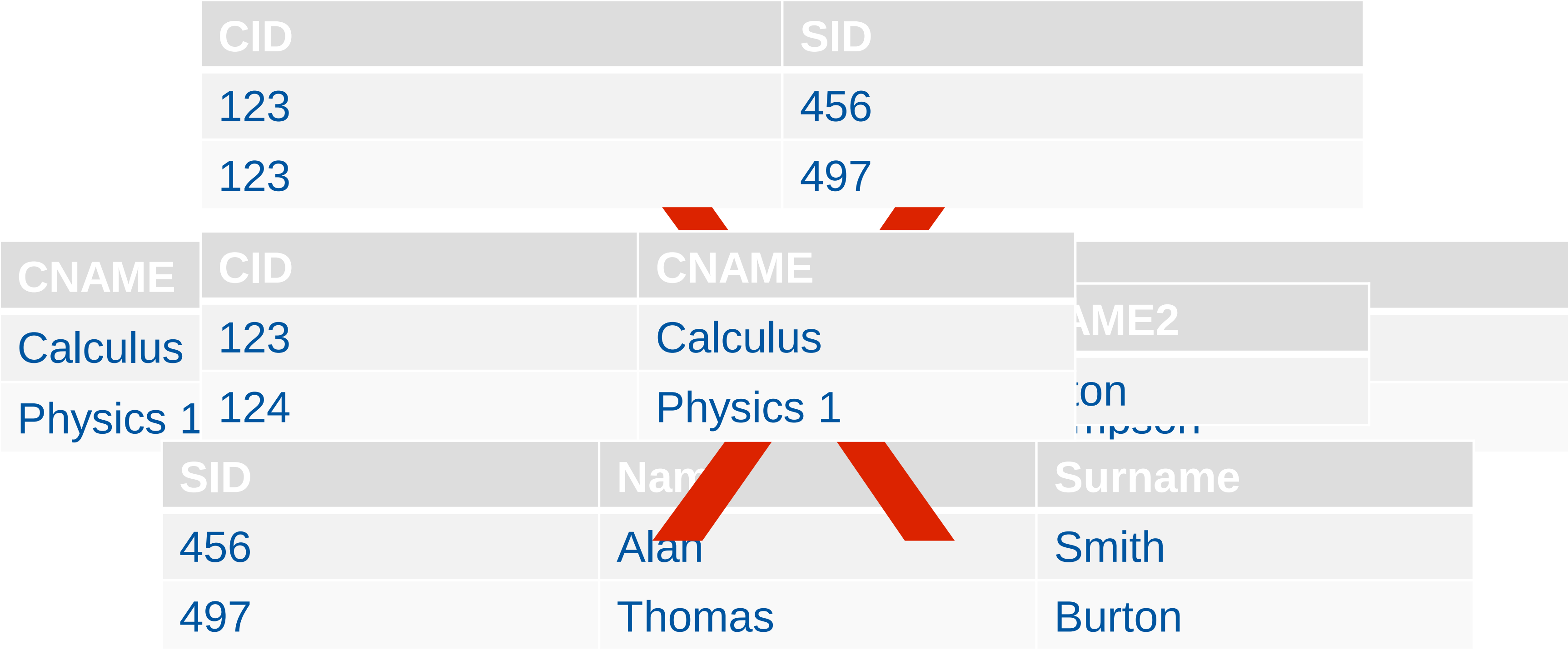


Normalization

- Objective – validate and improve a logical design, satisfying constraints and avoiding duplication of data
- Normalization is a process of decomposing relations with anomalies to produce smaller well-structured tables:
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Other: Boyce/Codd Normal Form (BCNF), 4NF ...
- Usually the 3NF is appropriate for real-world applications

First Normal Form (1NF)

- All table attributes values must be atomic (multi-values not allowed)
  - Eliminate duplicative columns from the same table
  - Create separate tables for each group of related data and identify each row with a unique column (the primary key)



CID	SID
123	456
123	497

CNAME
Calculus
Physics 1

CID	CNAME
123	Calculus
124	Physics 1

AME2
ton

SID	Nam	Surname
456	Alan	Smith
497	Thomas	Burton

Second Normal Form (2NF)

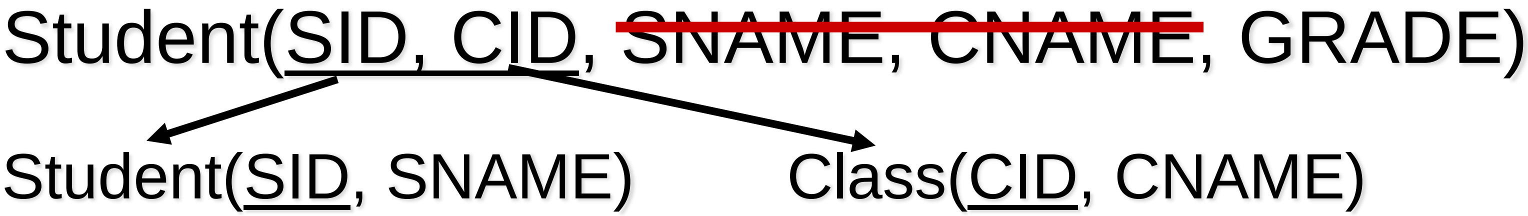
- 1NF
- No attribute is dependent on only part of the primary key, they must be dependent on the entire primary key

SID	SNAME	CID	CNAME	GRADE
456	Smith	123	Calculus	A
456	Smith	221	Physics	B
456	Smith	222	Database Management	B
497	Burton	123	Calculus	A
497	Burton	127	OO Programming	A
497	Burton	222	Database Management	B

Violation of the 2NF!

Normalization to 2NF

- For each attribute in the primary key that is involved in partial dependency – create a new table
- All attributes that are partially dependent on that attribute should be moved to the new table



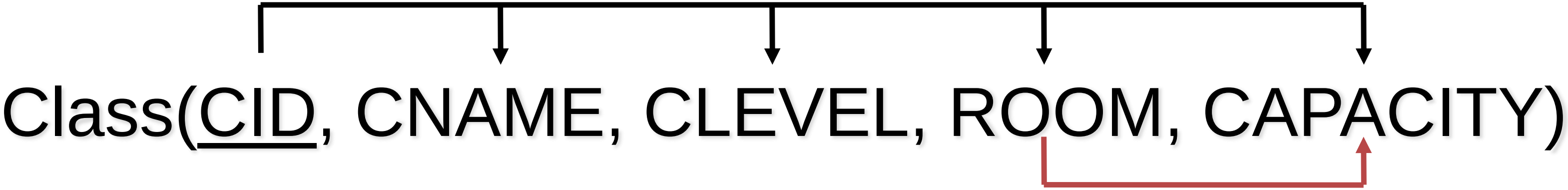
Student(SID, CID, ~~SNAME, CNAME~~, GRADE)

Student(SID, SNAME)

Class(CID, CNAME)

Third Normal Form (3NF)

- 2NF
- No transitive dependency for non-key attributes
  - Any non-key attribute cannot be dependent on another non-key attribute

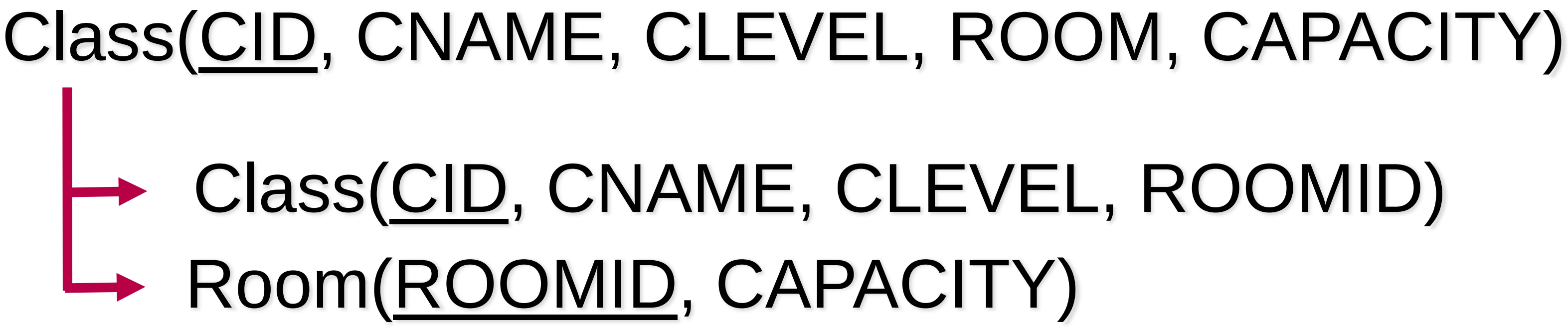


Class(<u>CID</u>, CNAME, CLEVEL, ROOM, CAPACITY)

Violation of the 3NF!

Normalization to 3NF

- For each non-key attribute that is transitive dependent on a non-key attribute, create a table



Class(CID, CNAME, CLEVEL, ROOM, CAPACITY)

Class(CID, CNAME, CLEVEL, ROOMID)

Room(ROOMID, CAPACITY)

Agenda

- **Database design**
    - Schema design
    - **Integrity constraints**
    - Best practices
- PL/SQL tips and tricks
- Robust application design

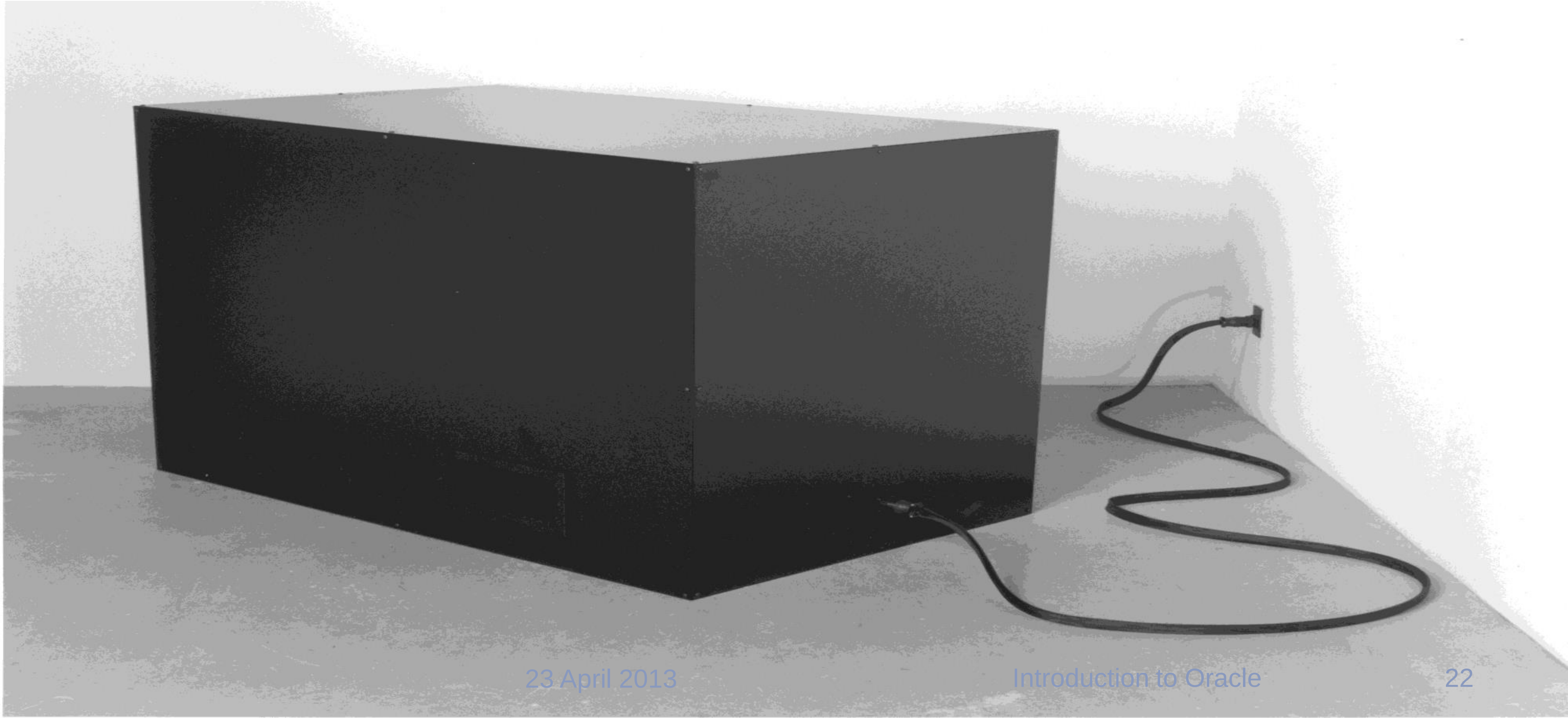
Primary keys

- Role: Enforce entity integrity
- Attribute or set of attributes that uniquely identifies an entity instance
- Every entity in the data model must have a primary key that:
  - is a non-null value
  - is unique
  - it does not change or become null during the table life time (time invariant)
- Use the shortest possible types for PK columns

Foreign keys

- Role: maintains consistency between two tables in a relation
- The foreign key must have a value that matches a primary key in the other table or be null
- An attribute in a table that serves as primary key of another table
- Use foreign keys!
  - foreign keys with indexes on them improve performance of selects, but also inserts, updates and deletes
  - indexes on foreign keys prevent locks on child tables

Not the best approach



Integrity Checks

- Use DB enforced integrity checks
  - Blindingly fast
  - Foolproof
  - Increases system self-documentation
- NOT NULL
- Client side integrity checks
  - Not a substitute for server side checks
  - Better user experience
  - Pre-validation reduces resource usage on server

Agenda

- **Database design**
  - Schema design
  - Integrity constraints
  - **Best practices**
- PL/SQL tips and tricks
- Robust application design

Schema design

- Column types and sizing columns
  - VARCHAR2(4000) is not the universal column type
    - high memory usage on the client
    - it makes data dump, not database
    - use proper data types, it:
      - Increases integrity
      - Increases performance
      - Might decrease storage needs (IO is time)
  - Put “nullable” columns at the end of the table

Schema design

- Estimate future workload
  - Read intensive?
  - Write intensive?
  - Transaction intensive?
  - Mixture? – estimate the amount of each type
- Design indexes knowing the workload
  - What will users query for?
    - Minimize number of indexes using proper column order in the indexes – use multicolumn indexes
    - Create views, stored procedures (PL/SQL) to retrieve the data in the most efficient way – easier to tune in a running system
  - What is the update/insert/delete pattern?
    - Create indexes on foreign keys

Indexes

- Less known but worth mentioning:
  - Local indexes vs global indexes
    - Local indexes
      - Stay valid through partition exchange
      - If not prefixed with partition key columns each partition must be searched
    - Global indexes
      - Can be ranged partitioned differently than table
      - Can enforce uniqueness
      - Range/interval partitioning only
  - Function based index/virtual column index
    - Built on function or complex calculation
      - *create index users_Idx on users (UPPER(name));*
        - Speeds up case insensitive searches
          - *select * from users where UPPER(name)=‘SMITH’;*

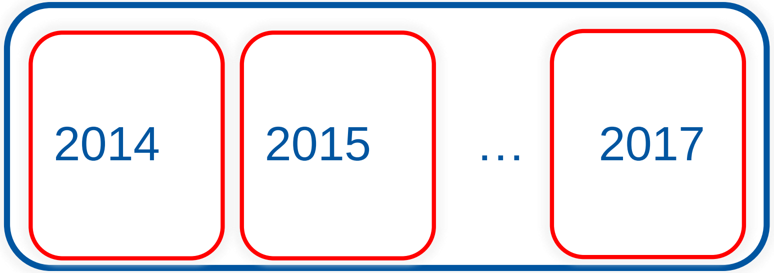
Partitioning

- Benefits:
  - Administration
    - Moving smaller objects if necessary, easier deletion of history, easier online operations on data
  - Performance
    - Use of local and global indexes, less contention in RAC environment
    - Partition pruning – scanning only needed partitions

Interval partitioning

- Automatic partition creation
  - Only 1st partition created manually
- Based on dates and number
- Extension of range partitioning
  - Migration to intervals is advised

Table jobs with execution date



2014

2015

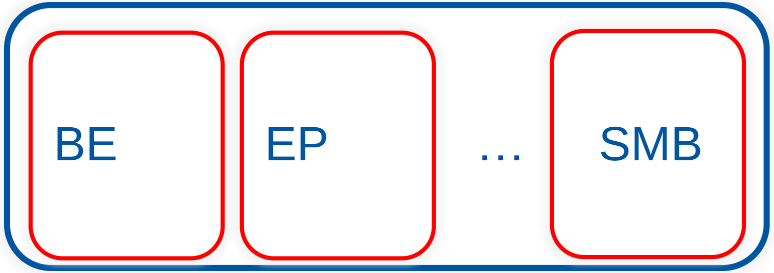
…

2017

List partitioning

- Based on a discrete value
- Check constraint on the key column is advisable
- In Oracle 11g requires manual partition creation
  - Automatic creation in 12c

Table employee with department



Existing tables vs partitoning

- In 12c simple and online
  - *alter table …modify partition by range..*
- In 11g
  - Still possible
    - Using *dbms_redefinition* set of commands
    - Last step requires application downtime
    - Contact your DBA ☺

IOTs

- Suppose we have an application retrieving documents uploaded by given users, list’s content and size are dynamic
  - In traditional table rows will be scattered, read index then data block
  - If the table was created as IOT:
    - *create table myIOT (…) organization index;*
    - Reads index blocks only
  - Also useful in:
    - Association tables in many to many relationships
    - Logging applications (parameter_id and timestamp as PK)

Compression

- Table compression
  - Reduces data size by 2 to 10 times
  - Simple compression
    - Only for direct inserts (archival, read only data)
      - *create table as select (…) compress;*
      - Insert append
  - Advanced compression
    - Works with read/write workloads
- Index compression
  - Simple, can vastly improve query performance
  - Low cardinality columns should only be compressed
  - Compression depends on selectivity
    - *create index employe_Idx on employees (deptID, groupId, supervisorID) (…) compress 1;*

Views

- Use views to simplify queries
- Don’t build up multiple view layers
  - Oracle optimizer might come up with suboptimal execution plan

Materialized views

- Materialized views are a way to
  - Snapshot precomputed and aggregated data
  - Improve performance
- Real-life example
  - Web page presenting a report
  - Multiple users accessing web page
  - Hundreds of request from the web server per second

  … try a materialized view to store that report
- RESULT_CACHE hint
  - Invalidated after DML on underlying objects
- Refresh your views only when needed
  - ‘on commit’ refreshes are very expensive

Denormalization

- Denormalized DB and Non-normalized DB are not the same thing
- Reasons against
  - Acceptable performance of normalized system
  - Unacceptable performance of denormalized system
  - Lower reliability
- Reasons for
  - No calculated values
  - Non-reproducible calculations
  - Multiple joins

Function based columns

Materialized views

Denormalization

- 1st step: Talk to your DBAs
- Main issues
  - Keeping redundant data correct
  - Identifying reasonable patterns
  - Correct order of operations
- Patterns
  - FETCH
    - Copy item’s price from ITEMS to ORDER_LINES
  - AGGREGATE
    - Put the order_price in ORDERS
  - EXTEND
    - Keep extended_price (price*quantity) in ORDER_LINES
- http://database-programmer.blogspot.com/2008/10/argument-for-denormalization.html

Agenda

- Database design
- **PL/SQL tips and tricks**
- Robust application design

PL/SQL – tips & tricks

- Query parse types
    - Hard parse
        - Optimizing execution plan of a query
        - High CPU consumption
    - Soft parse
        - Reusing previous execution plan
        - Low CPU consumption, faster execution
- Reduce the number of hard parses
    - Put top executed queries in PL/SQL packages/procedures/functions
    - Put most common queries in views
    - It also makes easier to tune bad queries in case of problems

PL/SQL – tips & tricks

- Reduce the number of hard parses
  - Use bind variables
    - Instead of:

```
select ... from users where user_id=12345
```

    - Use:

```
select ... from users where user_id=:uid
```

    - Using bind variables protects from sql injection

PL/SQL – tips & tricks

- Beware of bind variables peeking
  - Optimizer peeks at bind variable values before doing hard parse of a query, but only for the first time
  - Suppose we have huge table with jobs, most of them already processed (processed_flag = 'Y'):
    - using bind variable on processed_flag **may** change query behavior, depending on which query is processed first after DB startup (with bind variable set to 'Y' or 'N')
  - On a low cardinality column which distribution can significantly vary in time – do not use bind variable only if doing so will result in just a few different queries, otherwise **use bind variables**

PL/SQL – tips & tricks

- Use PL/SQL as an API
  - Provide abstraction layer
  - Make tuning easier
  - Restrict functionality
- Reduce the number of hard parses
  - Prepare once, execute many
    - Use prepared statements
    - Dynamic SQL executed thousands of times – consider *dbms_sql* package instead of *execute immediate*
    - Use bulk inserts whenever possible

PL/SQL – tips & tricks

- Stored procedures vs materialized views
  - Use SPs when refresh on each execution is needed
- Use fully qualified names
    - Instead of:

```
select ... from table1 ...
```

    - Use:

```
select ... from schema_name.table1 ...
```

  - Known bugs – execution in a wrong schema

Agenda

- Database design
- PL/SQL tips and tricks
- **Robust application design**

Writing robust applications

- Use different level of account privileges
  - Application owner (full DDL and DML)
  - Writer account (grant read/write rights to specific objects)
  - Reader account (grant read rights)
  - Directly grant object rights or use roles
    - Caution – roles are switched off in PL/SQL code, one must set them explicitly.

Writing robust applications

- Use connection pooling
  - Connect once and keep a specific number of connections to be used by several client threads (pconnect in OCI)
  - Test if the connection is still open before using it, otherwise try reconnecting
  - Log connection errors, it may help DBAs to resolve any potential connection issues

Writing robust applications

- Error logging and retrying
  - Trap errors
  - Check transactions for errors, try to repeat failed transactions, log any errors (including SQL that failed and application status – it might help to resolve the issue)
- Instrumentalization
  - Have ability to generate trace at will
  - More information in Performance Tuning talks

Writing robust applications

- Design, test, design, test ...
- Try to prepare a testbed system – workload generators, etc.
- Do not test changes on a live production system
- IT-DB provides test and integration system (preproduction) with the same Oracle setup as on production clusters
  - contact Oracle.Support to obtain accounts and ask for imports/exports

Questions?