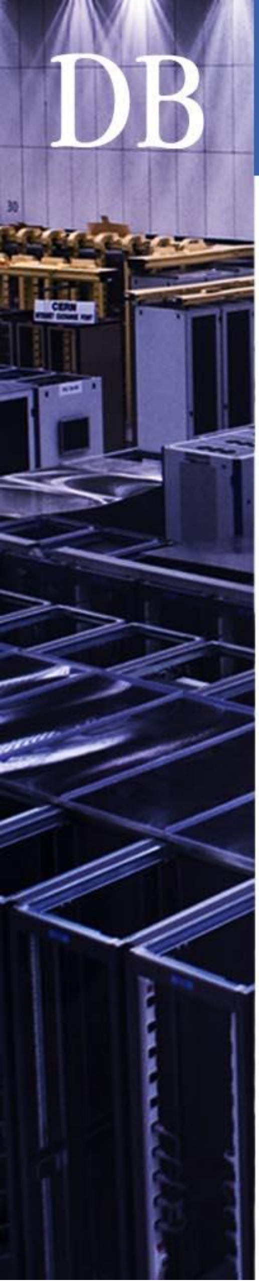
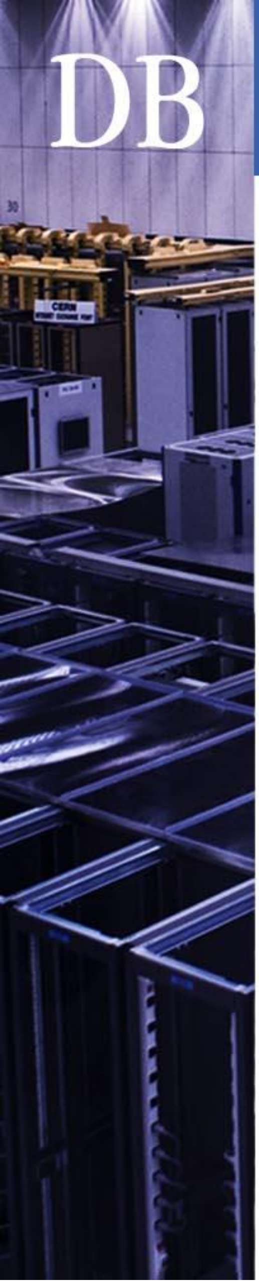


Database and Application Design

Katarzyna Dziedziniewicz-Wojcik

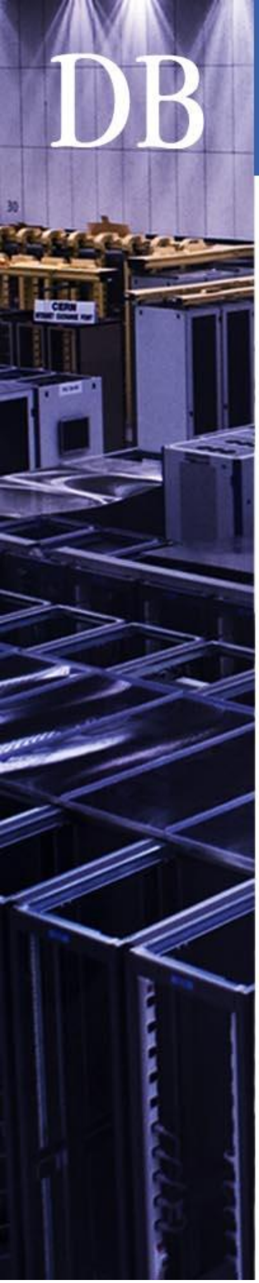
IT-DB

DB Design based on slides by Dawid Wojcik



Outline

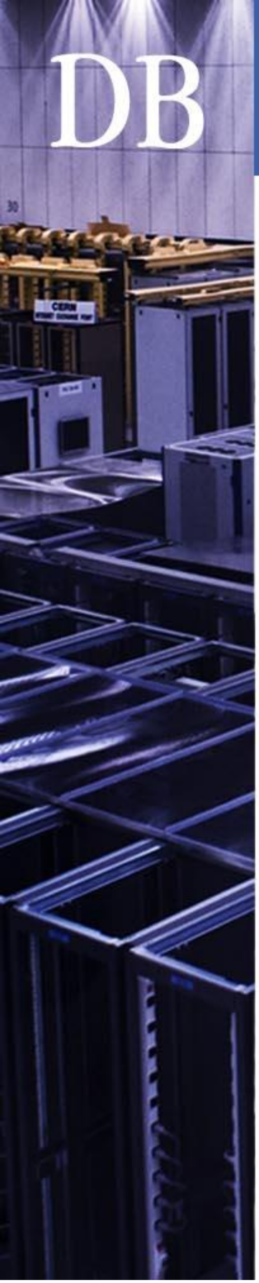
- Database design
- Tips & tricks
  - Indexes and Index Organized Tables
  - Views, Materialized Views
  - Partitioning
  - PL/SQL
- Writing robust applications
- Q&A



Database design

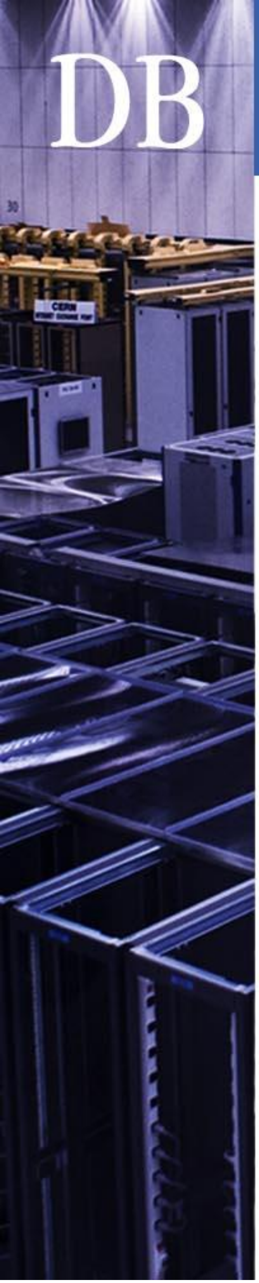
“It’s a Database, not a Data Dump”

- 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!



Database design - goals

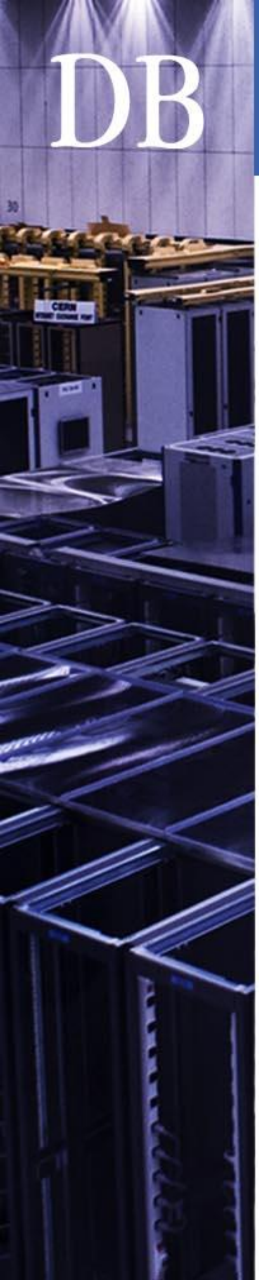
- Database design – define how to store data to:
  - avoid unnecessary redundancy
    - Storage is not unlimited.
    - Redundant data is not logically related
  - retrieve information easily and efficiently
    - Easily – does not necessarily mean with a simple query.
    - Efficiently – using built-in 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

- 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



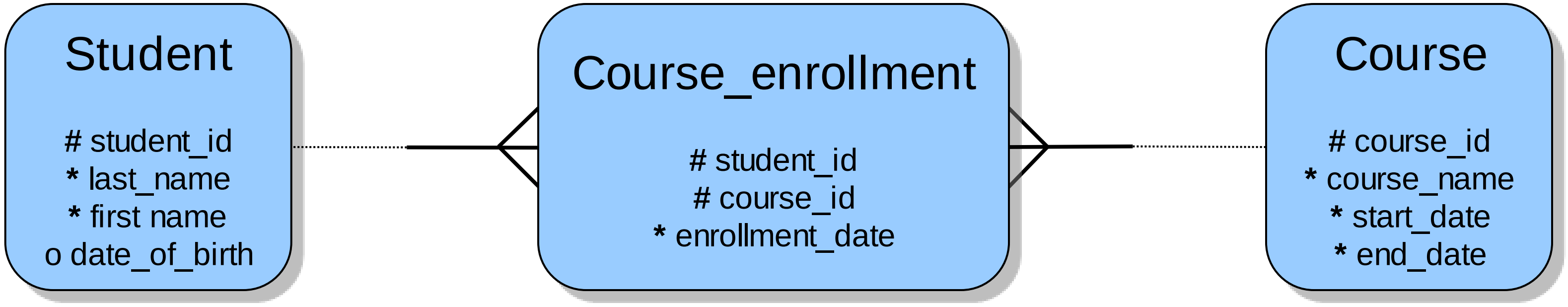
Conceptual design – practice

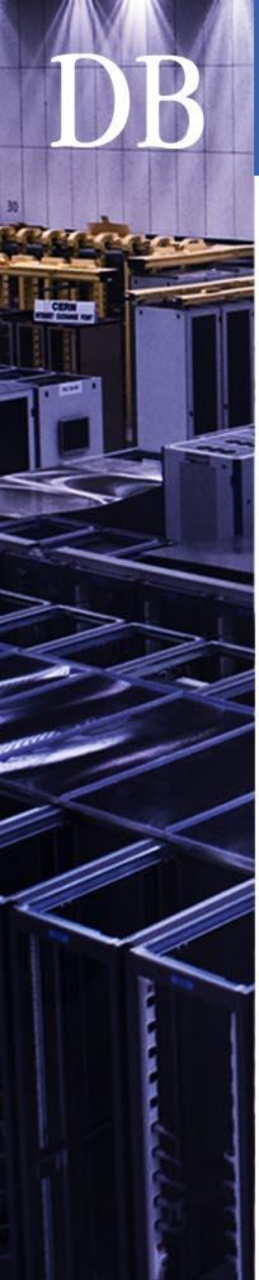
- The Entity-Relationship model (ER) is most common conceptual model for database design:
  - Describes the data in a system and how data is related.
  - Describes data as entities, attributes, and relationships.
  - Can be easily translated into many database implementations.

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)

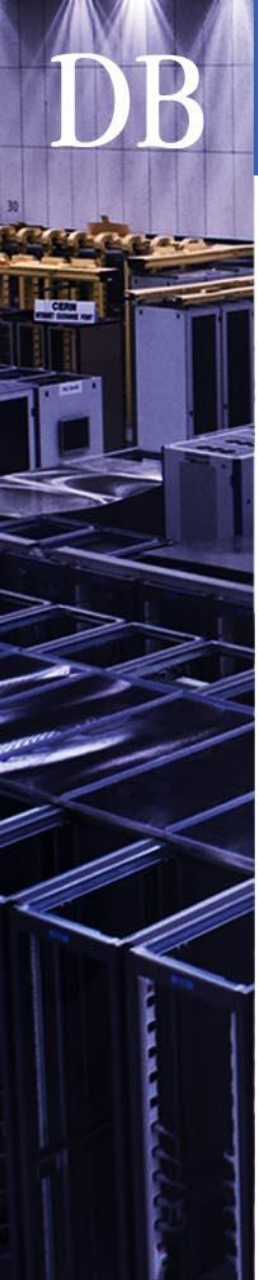
- All table attributes values must be atomic (multi-values not allowed)
  - Eliminate duplicative columns from the same table.
  - Create separate tables for each group of related data and identify each row with a unique column (the primary key).

Manager	Subordinates
Helen Smith	John Doe, Marc Brown

Manager	Subordinate1	Subordinate2
Helen Smith	John Doe	Marc Brown

Manager ID	Subordinate ID
763	6
763	3

Employee ID	Name	Surname
3	Marc	Brown
6	John	Doe
763	Helen	Smith

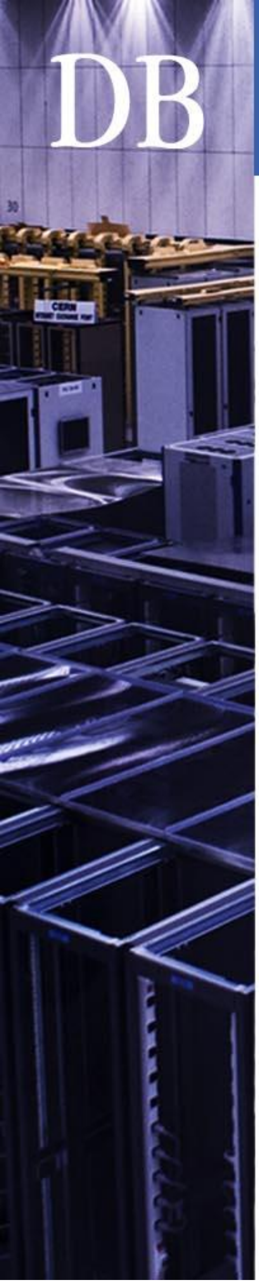


Second Normal Form (2NF)

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

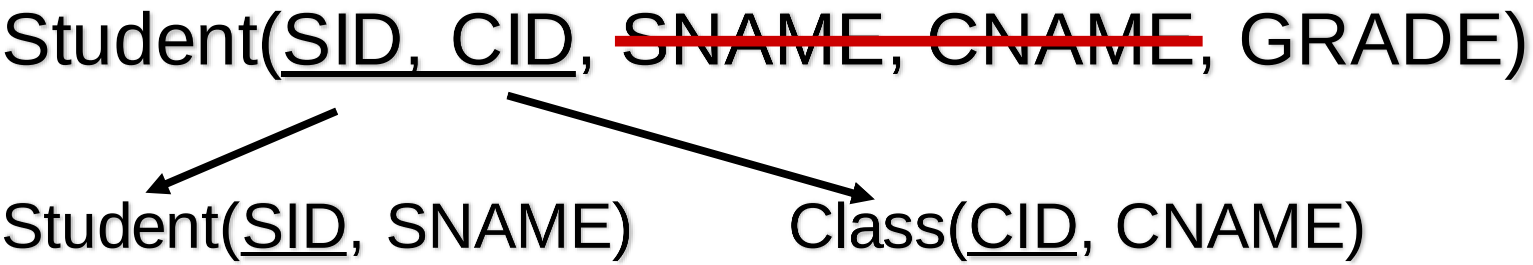
SID	SNAME	CID	CNAME	GRADE
224	Waters	M120	Database Management	A
224	Waters	M122	Software Engineering	B
224	Waters	M126	OO Programming	B
421	Smith	M120	Database Management	B
421	Smith	M122	Software Engineering	A
421	Smith	M125	Distributed Systems	B

Violation of the 2NF!



Normalization to 2NF

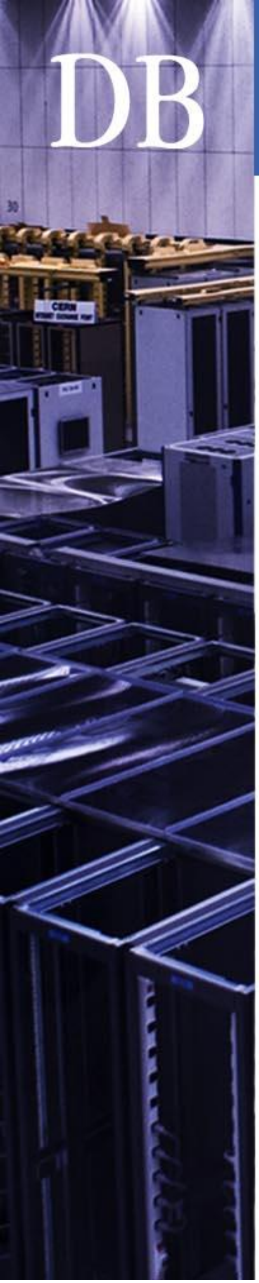
- For each attribute in the primary key that is involved in partial dependency – create a new table.
- All attributes that are partially dependent on that attribute should be moved to the new table.



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

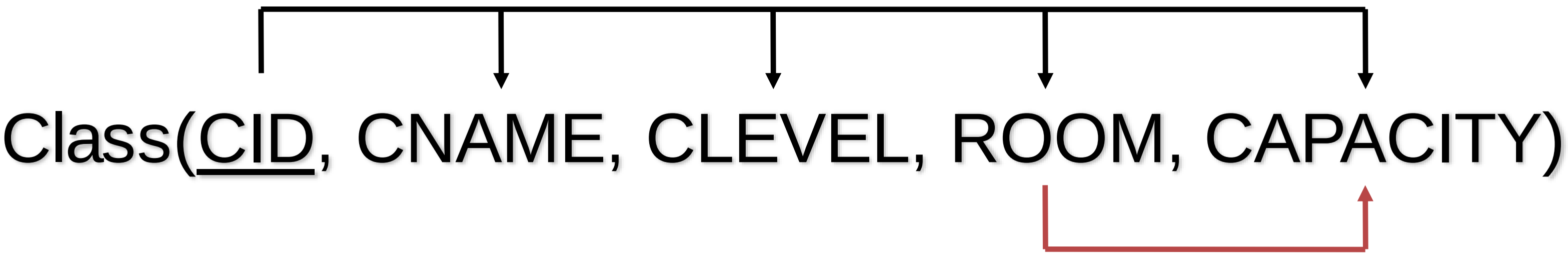
Student(SID, SNAME)

Class(CID, CNAME)

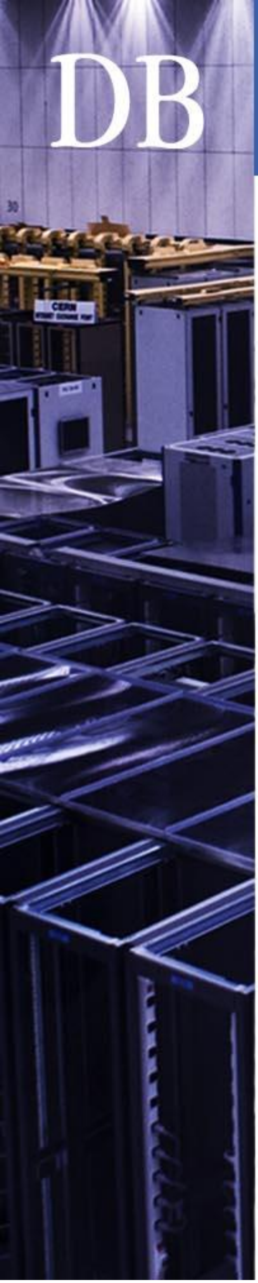


Third Normal Form (3NF)

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

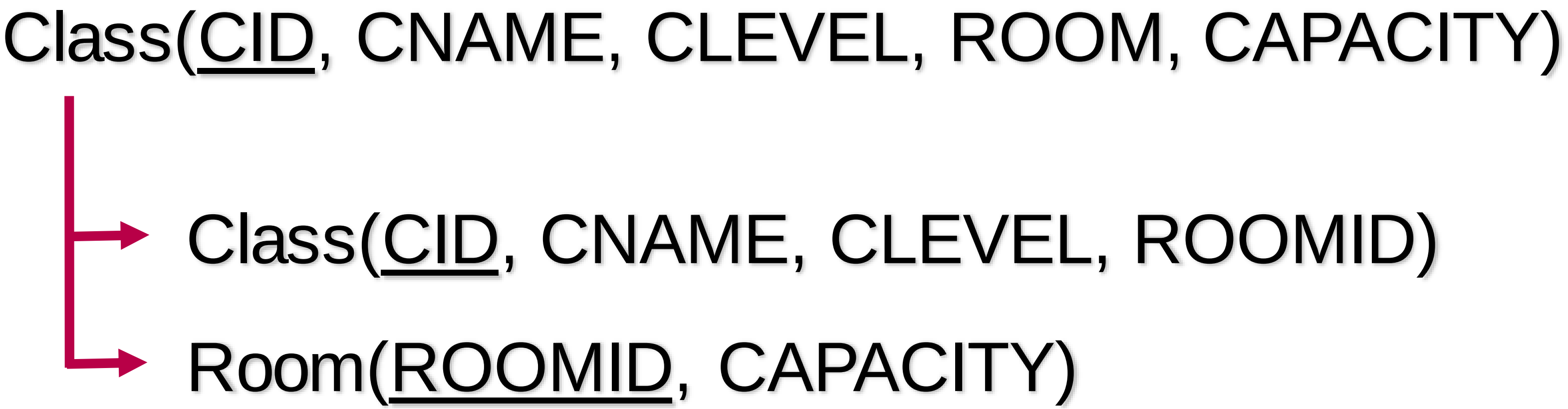

Class(CID, CNAME, CLEVEL, ROOM, CAPACITY)

Violation of the 3NF!



Normalization to 3NF

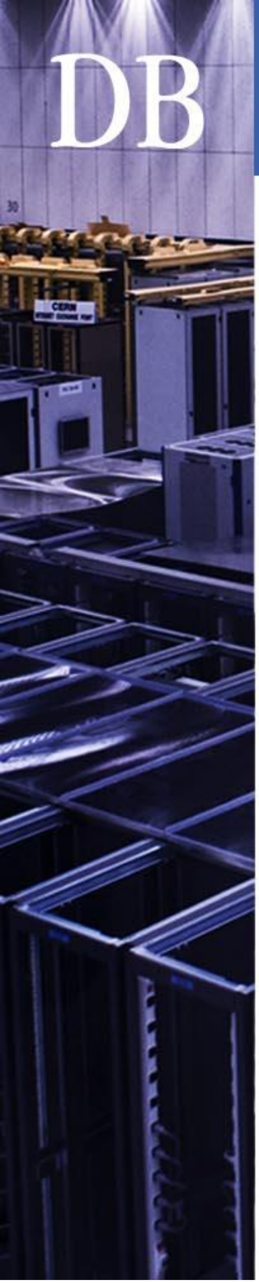
- For each non-key attribute that is transitive dependent on a non-key attribute, create a table.



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

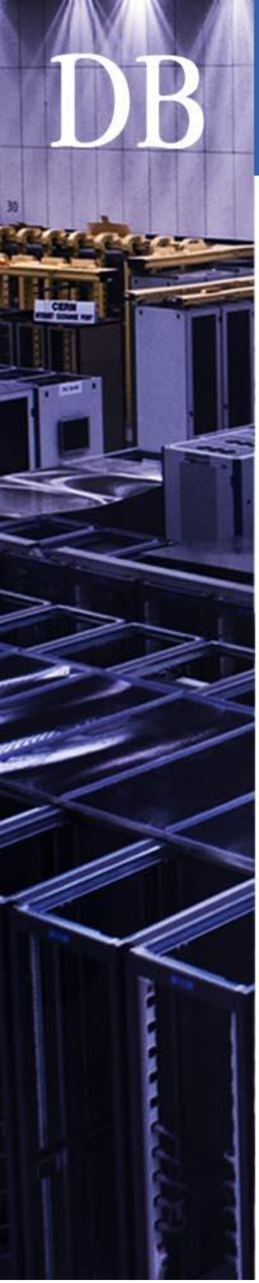
Class(CID, CNAME, CLEVEL, ROOMID)

Room(ROOMID, CAPACITY)



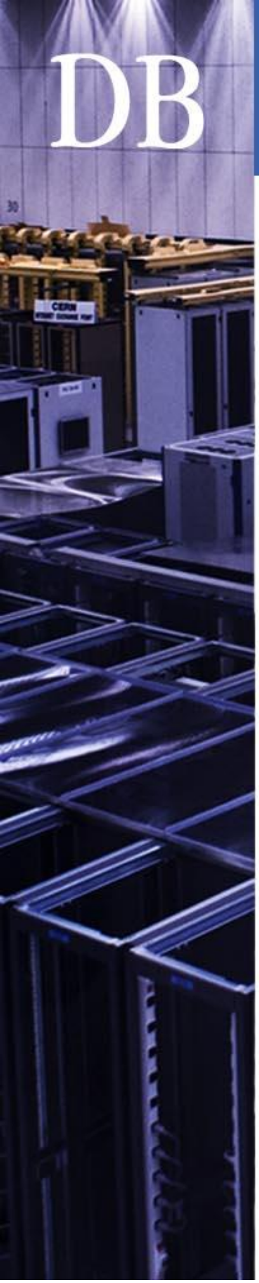
Integrity constraints - PK

- Primary keys (PK)
  - 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)



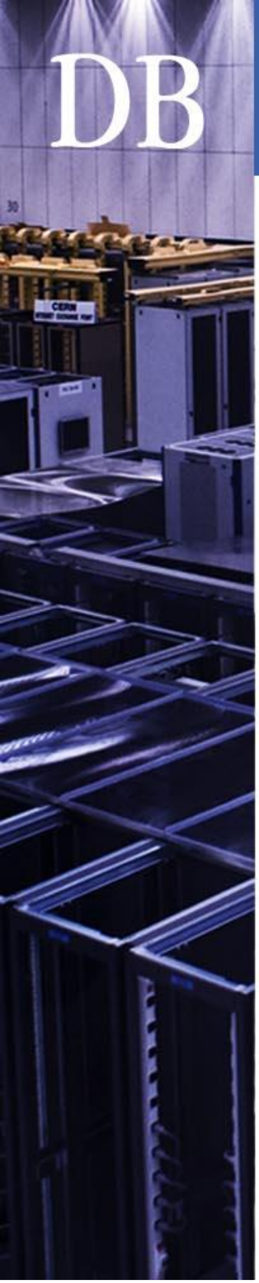
Integrity constraints - FK

- Foreign keys (FK)
  - Role: maintains consistency between two tables with 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.



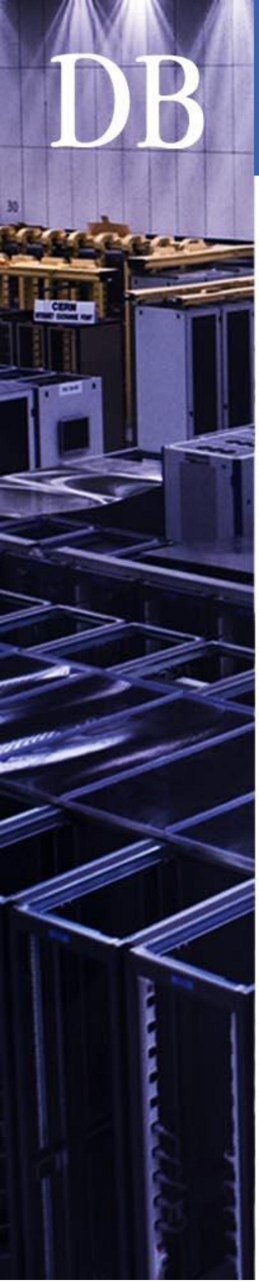
Integrity Checks

- Use DB enforced integrity checks
  - Blindingly fast
  - Proof to compromising
  - Increases system self-documentation
- NOT NULL
- Client side integrity checks
  - Not a substitute for server side checks
  - Better user experience
  - Reduces resource usage on server



Schema design – best practices

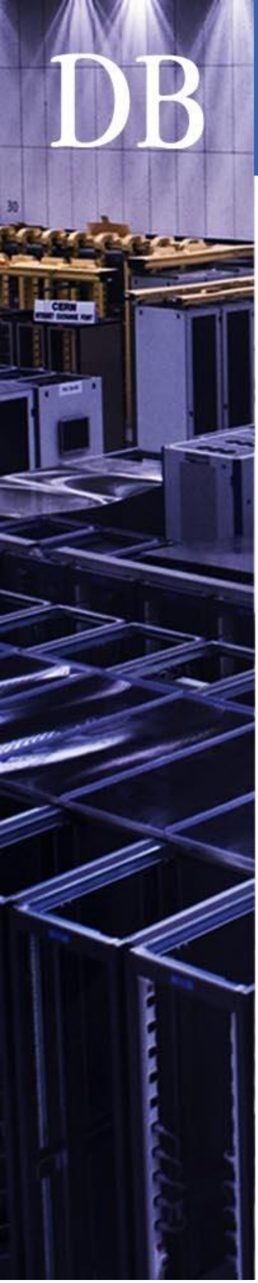
- Column types and sizing columns
  - VARCHAR2(4000) is not the universal column type
    - high memory usage on the client
    - it makes data dump, not database
    - use proper data types, it:
      - Increases integrity
      - Increases performance
      - Might decrease storage needs
  - Put “nullable”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



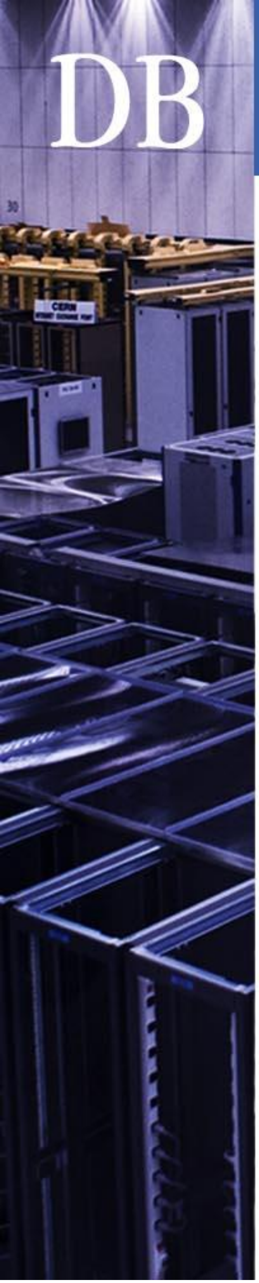
Schema design – best practices

- 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.
    - 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

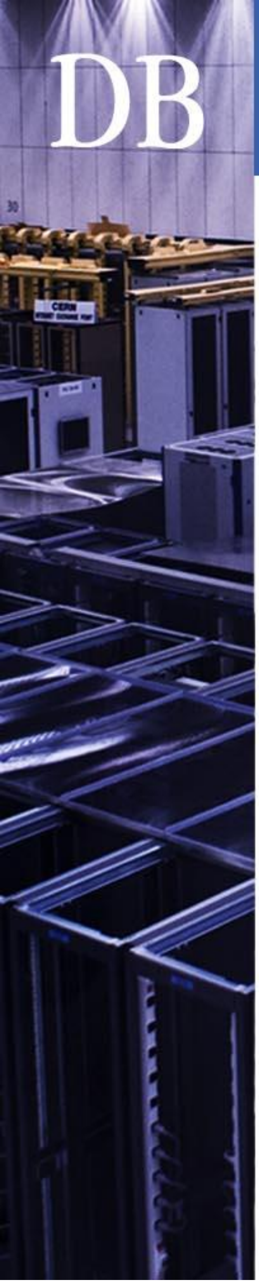


- Less known but worth mentioning:
  - Reversed index
    - Should be used when PK is populated by an increasing sequence
      - Decreases contention on index (especially important in RAC environment)
      - Cannot be used for range scans
  - Function based index/virtual column index
    - Built on function or complex calculation
      - For example on UPPER(NAME)
        - » Speeds up case insensitive searches



IOTs

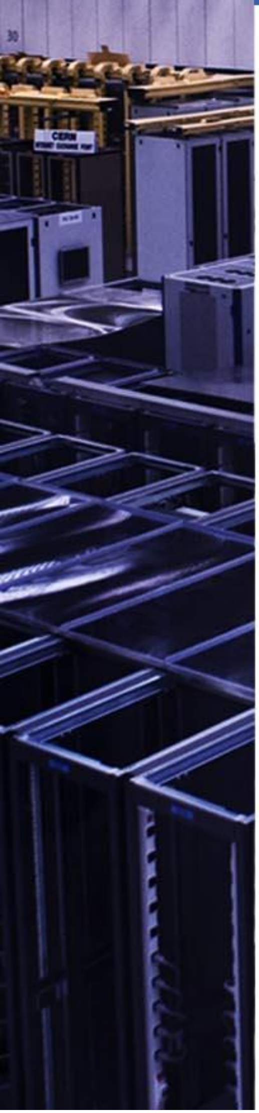
- Suppose we have an application retrieving documents uploaded by given users. List’s content and size are dynamic.
  - In traditional table rows will be scattered, read index then data block
  - If the table was created as IOT:
    - *create table myIOT (…) organization index;*
    - Reads index blocks only
  - Also useful in:
    - Association tables in many2many relationships
    - Logging applications (parameter_id and timestamp as PK)



Views

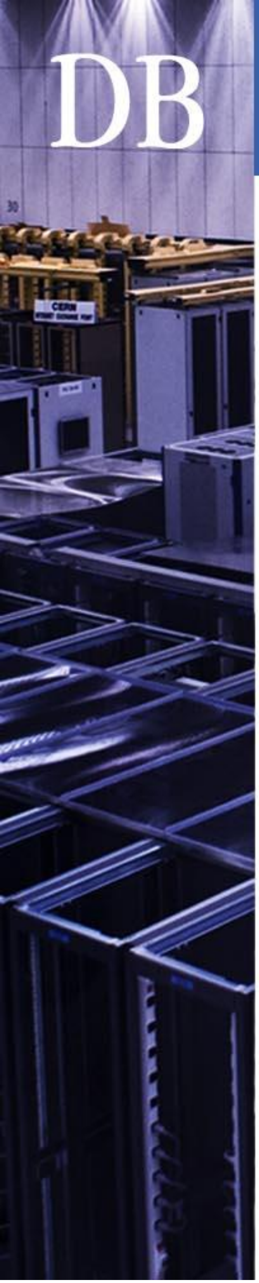
- Use views to simplify queries
- Don’t build up multiple view layers
  - Oracle optimizer might come up with suboptimal execution plan

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

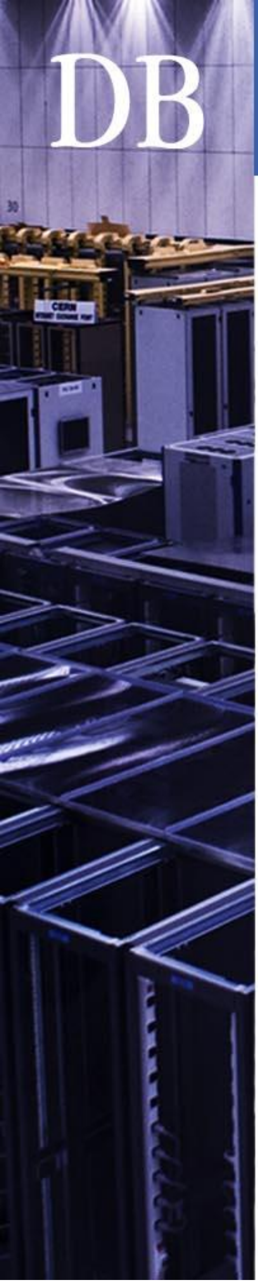
  … try a materialized view to store that report
- RESULT_CACHE hint
  - Invalidated after DML on underlying objects



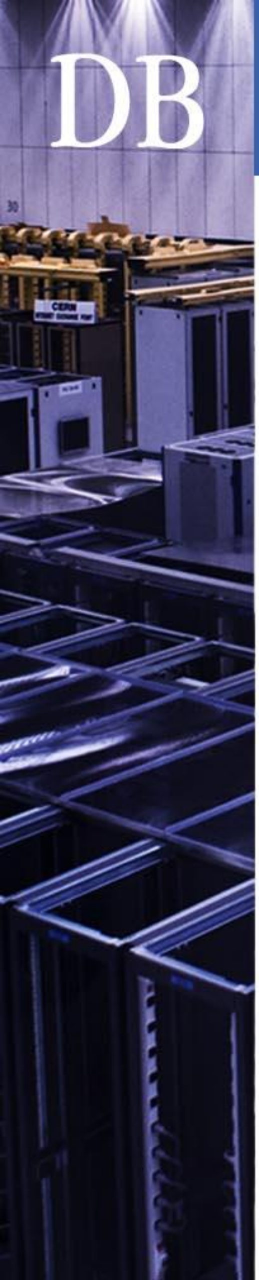
Partitioning – tips & tricks

- Investigate partitioning your application
  - You can try partitioning by time, subdetector, subsytem, etc.
    - Interval partitioning now available in Oracle
  - Benefits:
    - increased availability – in case of loosing one tablespace/partition,
    - easier administration – moving smaller objects if necessary, easier deletion of history, easier online operations on data
    - increased performance – use of local and global indexes, less contention in RAC environment.

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

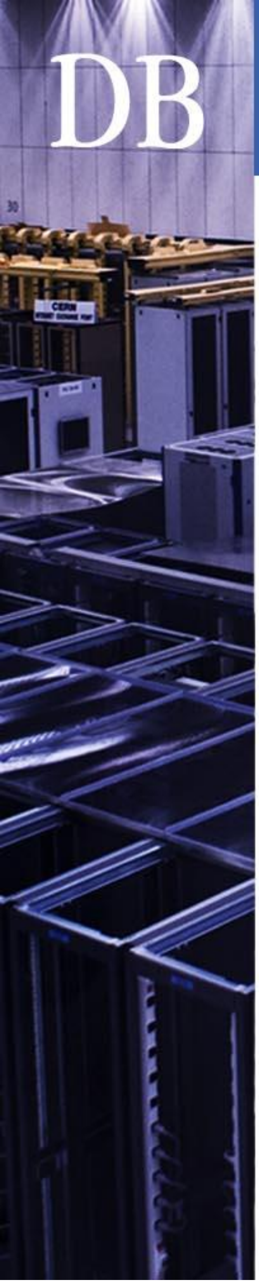
- Reduce the number of hard parses
  - Use bind variables
    - Instead of:

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

    - Use:

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

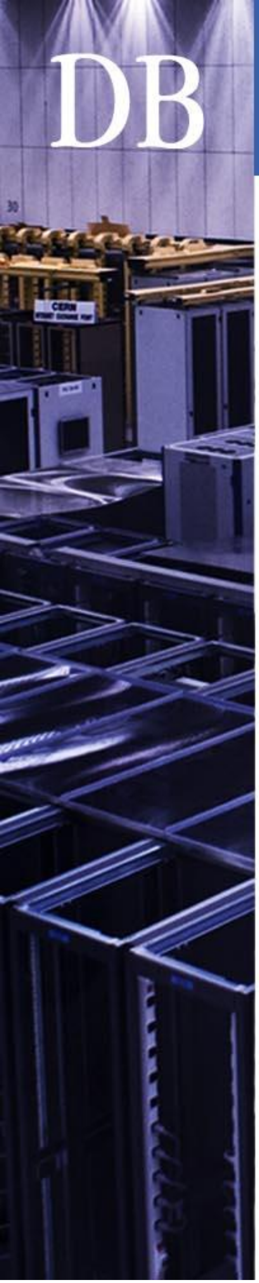
    - Using bind variables protects from sql injection
      - More on SQL injection in Szymon’s talk



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 **<u>may</u>** 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 **<u>use bind variables</u>**.

PL/SQL – tips & tricks



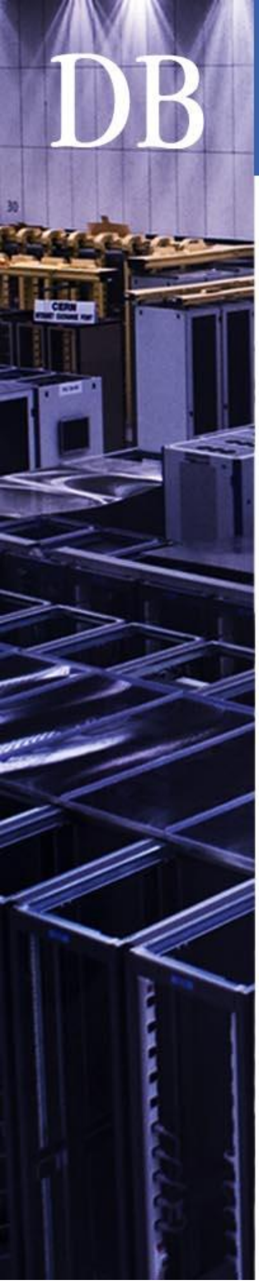
- Reduce the number of hard parses
  - Prepare once, execute many
    - Use prepared statements
    - Dynamic SQL executed thousands of times – consider *dbms_sql* package instead of *execute immediate*
    - Use bulk inserts whenever possible
- Use fully qualified names
    - Instead of:

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

    - Use:

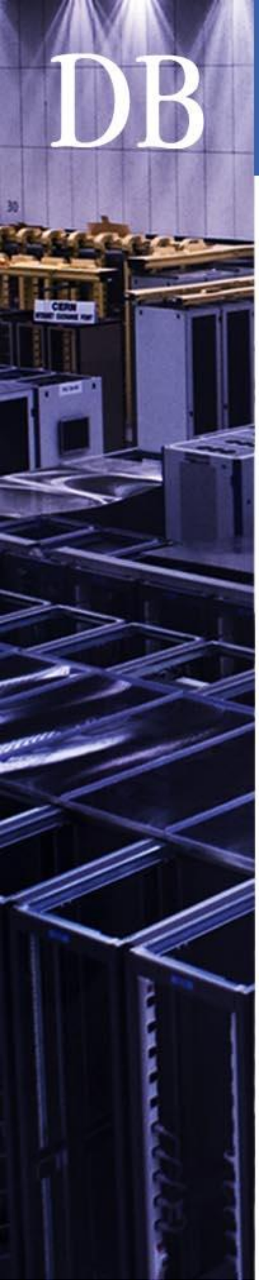
```
select ... from schema_name.table1 ...
```

  - Known bugs – execution in a wrong schema



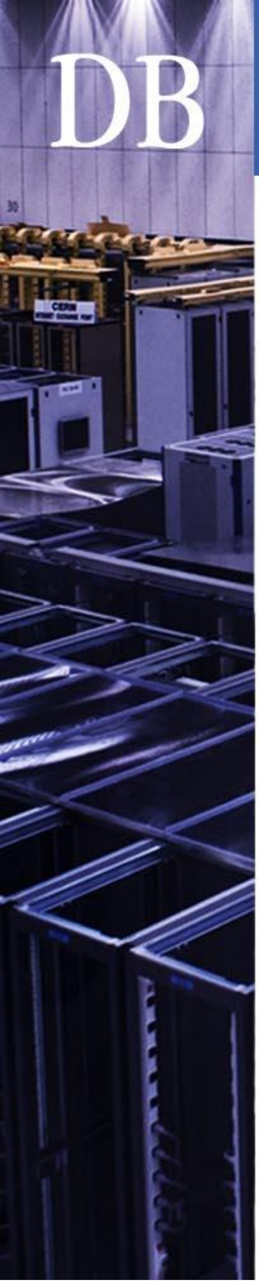
Writing robust applications

- 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.
  - – More on security in Daniel’s talk



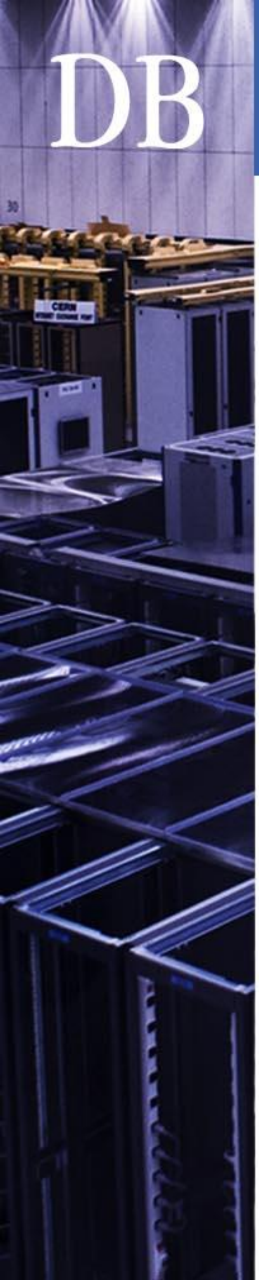
Writing robust applications

- 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



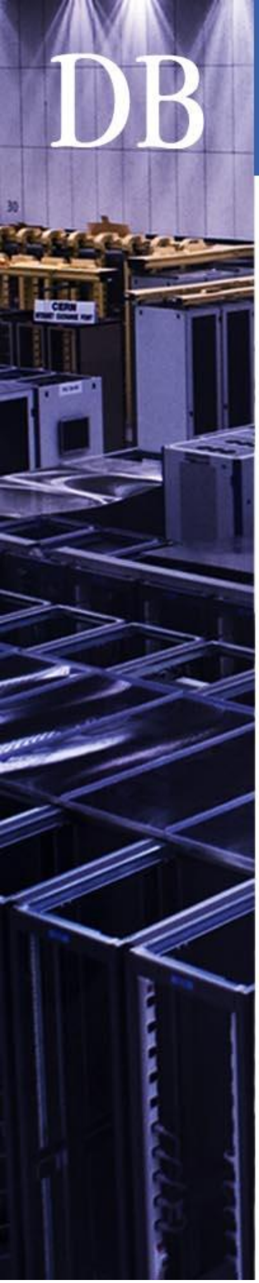
Writing robust applications

- 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 Chris'es talk



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.



Q & A