Topic 07: Physical Database Design

ICT285 Databases

Dr Danny Toohey

About this topic

In this topic, we complete our set of topics on the database design process, by looking at physical design – where we define exactly *how* we will implement the database in the target DBMS. In the labs, we'll return to Oracle to look in more detail at the decisions that need to be made and implemented during the physical database design phase.

Topic learning outcomes

After completing this topic you should be able to:

- Describe the activities in physical database design
- Design tables and integrity constraints for a target DBMS, based on the logical design
- Explain what views are and what they can be used for
- Draw a transaction usage map (TUM) for the most important transactions on a database
- Determine when denormalisation may be appropriate
- Use the SQL GRANT and REVOKE statements to manage user access to database objects
- Briefly describe other measures for DBMS-level security
- Document the physical database design

Resources for this topic

READING

- Text, Chapter 6: section on Data Types
- Text, Chapter 7: SQL for Database Construction and Application Processing (to p371, down to "Embedding SQL in Program Code"). We have covered much of this already in Topic 3 and the labs.
- Text, Chapter 9: section on Database Security p472-477

Other resources:

- Oracle documentation at https://docs.oracle.com/database/121/CNCPT/intro.htm#CNCPT001 (much more than you need!)

In this week's lab we return to Oracle as we start to look aspects of physical database design and implementation. We'll consider the data types available in Oracle, views, and access permissions using GRANT.

LAB 7 – PHYSICAL DATABASE DESIGN IN ORACLE

Topic outline

1. Where physical design fits into database design
2. Translate logical data model for target DBMS
3. Views
4. Analyse database usage
5. Denormalisation
6. Database security

1. Where physical design fits into database design

Reminder: Database Design

Process of creating a design for a database that will support the enterprise's mission statement and mission objectives for the required database system

Three phases of database design:

- Conceptual database design
- Logical database design
- Physical database design

Reminder (topic 5)

Conceptual Database Design (this topic)

- Process of constructing a model of the data used in an enterprise, independent of *all* physical considerations
- Data model is built using the information in users' requirements specification.
- Conceptual data model is source of information for logical design phase.
- Conceptual data models can be drawn us Entity-Relationship diagrams, UML, or oth modelling techniques

Logical and Physical Database Design (looking ahead to topics 6,7)

Logical design

- Process of constructing a model of the data used in an enterprise based on a specific data model (e.g. relational), but independent of a particular DBMS and other physical considerations.
- Conceptual data model is refined and mapped on to a logical data model.

Physical design

- Process of producing a description of the database implementation in secondary storage.
- Describes base relations, file organizations, and indexes used to achieve efficient access to data. Also describes any associated integrity constraints and security measures.
- Tailored to a specific DBMS system

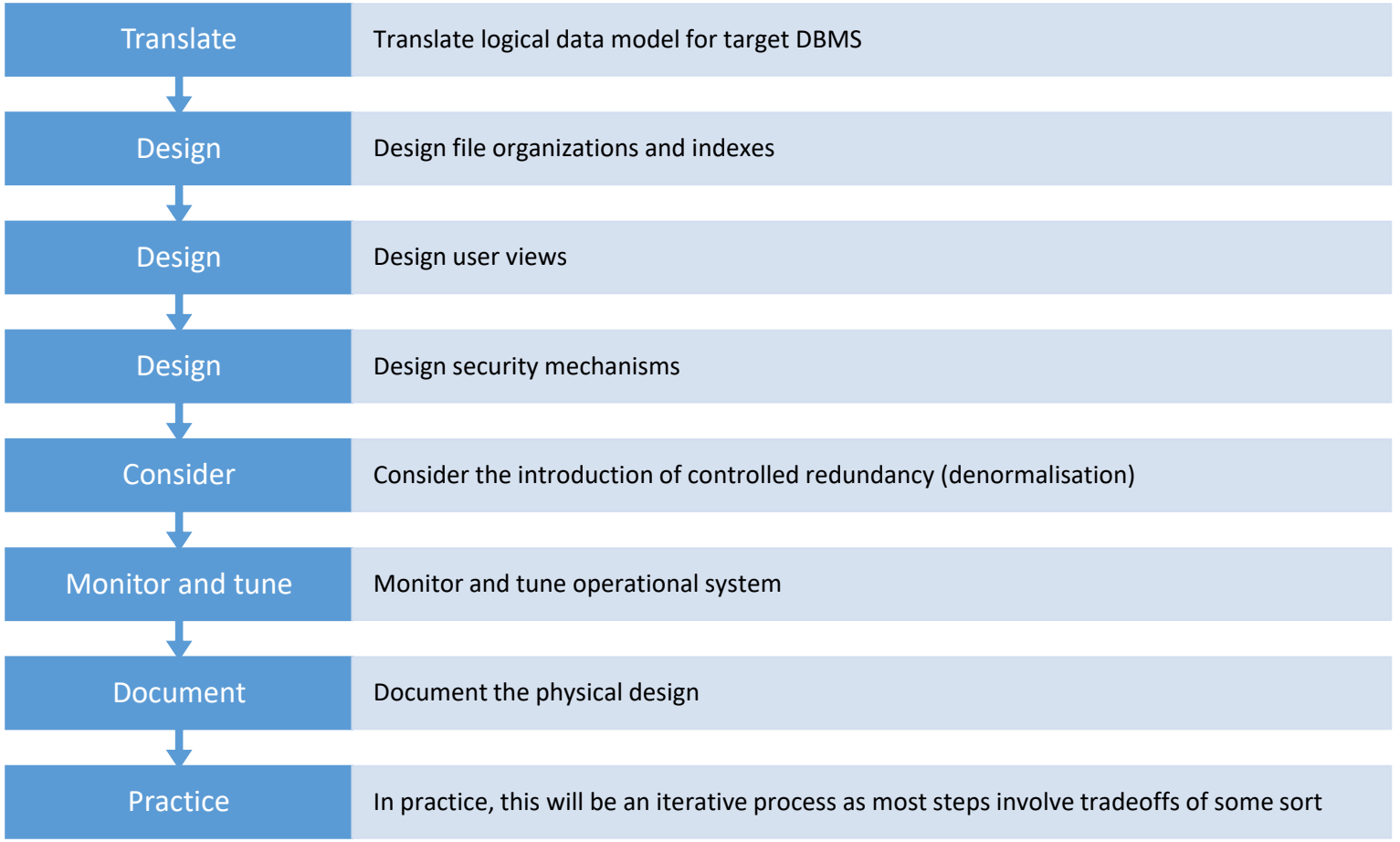
Physical database design

- Process of using data from logical design to create a description of the implementation of the database on secondary storage
- Describes file organisations, base relations, and indexes (FBI) used to achieve efficient data access. Also outlines any relevant security mechanism and integrity constraints
- Tailored to a specific DBMS system

- ~~The physical design process is where we concentrate on the **efficient implementation of our logical design**~~
- ~~It is important to note that efficiency for one operation often comes at the expense of another (e.g., retrieval vs update)~~

A physical database design methodology - steps

- *Steps from Connolly & Begg*



A physical database design methodology - steps

1. Translate logical data model for target DBMS
2. Design file organizations and indexes
3. Design user views
4. Design security mechanisms
5. Consider the introduction of controlled redundancy (denormalisation)
6. Monitor and tune the operational system
7. Document the physical design

- This topic
- Next topic

Steps from Connolly & Begg

2. Translate logical design to physical design in target DBMS

Translate logical design to physical design in target DBMS

- Choose the DBMS
- Design base tables
- Design integrity constraints
- Design representation of derived data

First, select a DBMS

Goal is to pick a system that allows expansion and allows faster, easier software building balanced against costs, that fulfils the requirements of the system

Need to:

- Analyse requirements of the user ~~(done as part of the system development process)~~
- Flush out scope and objective of research –
  - what criteria will be used for evaluation, etc
- Find potential products
- From the criteria shortlist products
- Evaluate products more deeply –
  - vendor demonstrations, benchmarking tests etc
- Advise selection

Can use **evaluation matrix** to compare on weighted criteria (e.g. see ICT284)

Types of selection criteria

"**Development considerations:** includes the data model, query functionality, available drivers, data consistency. These factors dictate the functionality of your application, and how quickly you can build it.

"Operational considerations: performance and scalability, high availability, data centre awareness, security, management and backups. Over the application's lifetime, operational costs will contribute a significant percentage to the project's Total Cost of Ownership (TCO), and so these factors constitute your ability to meet SLAs while minimizing administrative overhead.

"Commercial considerations: licensing, pricing and support. You need to know that the database you choose is available in a way that is aligned with how you do business."

(quoted directly from article by Mat Keep, 2015: https://www.mongodb.com/blog/post/introducing-database-selection-matrix)

Example evaluation matrix

Category	Problem	Importance	MySQL	Oracle	PostgreSQL
Elementary features	Basic data types	C	B	C	A
	SQL	B	B	B	B
	Declarative constraints	B	C	A	A
	Programming abstractions	A	C	A	C
Transactions	Transactions	A	D	A	A
	Locks	A	D	A	A
Programming in DB	Multiuser access	A	C	A	C
	Stored procedures and triggers	B	C	A	A
Administration	Access control	B	A	A	B
	Backup	A	C	A	C
	Data migration	C	A	B	A
Portability and	Portability	B	B	A	B
Scalability	Scalability	A	B	A	C
	Query optimization	A	B	A	B
	Structures supporting optimization	B	D	A	B
	Support for OLAP	B	D	A	D
Performance and VLDB	Allocation of the disk space	A	C	A	C
(Very Large DB)	Size limits	A	B	A	C
	VLDB implementation	A	D	A	B
	Access to multiple databases	C	C	A	C
Special data types	Large objects	B	B	A	C
	Post-relational extensions	C	D	A	B
	Support for special data types	C	D	A	C
Application development	Embedded SQL	C	D	A	B
and interfaces	Standard interfaces	B	B	A	B
	Additional interfaces	A	A	A	A
	Web technology	A	B	A	B
	XML	B	D	A	D
	CASE	B	D	A	D
Reliability	Recovery	A	C	A	C
Commercial issues	Prices	C	A	D	A
	Technical support	A	C	B	C
	Position in the market	A	C	A	C

Oracle8i and Oracle9i features are considered for the following comparisons, and MySQL versions base 4 and higher, PostgreSQL versions 7.2.x and higher.

From http://www-css.fnal.gov/dsg/external/freeware/mysql-vs-pgsql.html (link now dead)

Design base tables

~~In the **logical** design, we documented in the data dictionary:~~

~~For each relation:~~

- ~~the name of the table~~
- ~~a list of attributes~~
- ~~the PK and, where appropriate, FKs and alternate keys~~
- ~~referential integrity constraints for any FKs identified~~

~~For each attribute:~~

- ~~its domain, consisting of a data type, length, and any constraints on the domain~~
- ~~an optional default value for the attribute~~
- ~~whether it can have missing values~~
- ~~whether it is derived, and if so, how it should be computed~~

•The physical design should use the data dictionary from the logical design and map those requirements to the features available in the selected DBMS

Oracle data types

Include:

- VARCHAR2 (variable length)
- CHAR (fixed length character)
- NUMBER (including precision and scale)
- DATE
- … and many others

See text, 207-289 (p317-319 in 13th edition) and https://docs.oracle.com/database/121/CNCPT/tablecls.htm#CNCPT113

Choice of data type depends on suitability for attribute domain; extensibility; storage efficiency

VARCHAR2

- Use VARCHAR2 when character data of indeterminate length is required, e.g. names, addresses, etc
- Also when numeric data is used in an non-numeric way, e.g. phone numbers, credit card numbers
  - Will never need to do arithmetic on them
  - Can also store formatting information
  - Can filter records using LIKE, etc to select for regions
  - Expandable, e.g. to international numbers

CHAR

- CHAR always stores the same length field no matter the length of the contents – any unused space is padded with blanks
- This can have some benefits in storage efficiency (although not storage space) and simpler updating
- However VARCHAR is generally more flexible and more straightforward to deal with in queries and applications

NUMBER

- Use for numeric values that are used as numbers, rather than character strings
- Can define precision (total number of digits)
- And scale (number of digits to right of decimal)

12345.67

NUMBER(7,2)

- If scale is not specified, it is zero, e.g. NUMBER(2)
- Oracle doesn’t have INTEGER and will convert INTEGER to NUMBER(38)

DATE

- Stores point-in-time values as year, month, day, hours, minutes, seconds
- Standard format is DD-MON-YY, e.g. 19-SEP-17
- Can change format using TO_DATE function with format mask, e.g.

```
TO_DATE ('November 13, 1992', 'MONTH DD, YYYY')
```

- If any part of the date/time isn't specified the remainder defaults to e.g. midnight (00:00:00) if no time is entered; first day of current month and year if only time is entered

Other Oracle data types

- BLOB – stores unstructured binary data up to 128 terabytes (e.g. images, videos, sound)
- CLOB – stores up to 128TB of character data
- Floating point numbers can be stored with BINARY_FLOAT and BINARY_DOUBLE – use binary precision rather than decimal precision
- Note that Oracle **doesn't** support a Boolean data type - need to implement e.g. as character 'T' or 'F'; or numeric 1 or 0, etc

Implementing constraints

- The DBMS can determine what constraints can be implemented. The ways in which constraints can be implemented includes-
  - Through data definition features of the DBMS
  - Through SQL codes or triggers
  - In the application (through forms or in code)

- ~~As a general rule it is better to define a constraint with the database design – so it can be enforced consistently in every part of the application~~
- Better to implement a constraint with the database design (through DBMS definition) so it can be enforced on every part of application that access that database
- Forms or in application code tends to be more suited to implement complex constraints with flexible responses

Oracle integrity constraints

- NOT NULL
- UNIQUE Key
- PRIMARY KEY
- Referential Integrity constraints (FOREIGN KEY)
- CHECK integrity constraints

We have discussed these already in earlier topics and labs – refer back to them as you need to

See https://docs.oracle.com/database/121/CNCPT/datainte.htm#CNCPT021

Surrogate primary keys

- An artificial primary key created to simplify retrieval – e.g. if you have a very long concatenated candidate key
- Only used for implementation, usually created automatically by the DBMS

Advantages:

- Short, numeric, fixed – therefore 'good' primary key
- Simpler when used as foreign key in another table

Disadvantages:

- Meaningless to user
- May need further queries to retrieve 'meaningful' data
- May not be unique when multiple databases are merged

https://oracle-base.com/articles/12c/identity-columns-in-oracle-12cr1

Auto increment in Oracle

- Oracle 12c has an IDENTITY data type that acts as an Autonumber:

```
CREATE TABLE identity_test_tab (
  id          NUMBER GENERATED ALWAYS AS IDENTITY,
  description VARCHAR2(30) );

INSERT INTO identity_test_tab (description) VALUES ('Just DESCRIPTION');
```

	ID	DESCRIPTION
1	1	Just DESCRIPTION

Auto increment in earlier Oracle versions

- Oracle 11 and earlier do not have auto increment, but you can create a numerical *sequence* that will provide a sequential series of unique numbers to use as a surrogate key:

```
CREATE SEQUENCE seqCustomerID
INCREMENT BY 1 START WITH 100;
```

100, 101, 102, …

https://chartio.com/resources/tutorials/how-to-define-an-auto-increment-primary-key-in-oracle/

Referential actions in Oracle

- Recall from Topic 6 the different possibilities for maintaining referential integrity through updates and deletes to the parent table
- Of these, Oracle supports only:

  ON UPDATE NO ACTION **(default, do *not* specify in FK clause – *will produce an error*)**

  ON DELETE NO ACTION **(default, do *not* specify)**

  ON DELETE CASCADE (specify in FK clause)
- If other referential actions are required then they would have to be enforced in other ways, e.g. through triggers

Derived data

- Other attributes can be used to calculate a **derived attribute**, e.g.
  - **Profit** from the sale of a work of art
  - **Age** of individual artists
- A database can have a derived attribute (store)*:*
  - Have columns for ‘Profit’ and ‘Age’
- or *calculated every time* it is needed:
  - SalePrice–PurchasePrice
  - today–DateOfBirth
- How to decide which is appropriate?

Derived data: Store or Calculate? Issues

Store?

- Additional storage cost of the derived data
- Must re-calculate continually to keep it consistent with operational data it is derived from (e.g. use a trigger)

Calculate?

- Cost to calculate it each time it is required

Derived data: Store or Calculate? Considerations

- How often is it required?
- How hard is it to calculate?
  - Complex calculation?
  - Lots of tables involved?
- How frequently does it change?
  - If not storing, must recalculate derived value every time it changes
- Will additional storage be required much?

The take-aways…

- The logical design must be translated to the physical design using the target DBMS
- As each DBMS will have slightly different features, the final physical design will be DBMS-dependent
    - However, all of the logical design should be implemented
- Some design decisions include selecting appropriate data types, dealing with integrity constraints, and determining the best way to handle derived attributes

3. Views

Views

A view is the product of one or more relational operations such as select query that we are storing. The view essentially operates on the base tables to create another, 'virtual' table. They don't contain actual data unlike base tables

- User of the view are only able to view those parts of the database
- This restriction can be vertical (=PROJECT) or horizontal (=RESTRICT)

```
CREATE view emp_all
AS
SELECT first_name, last_name, middle_initial,
       street_address, state, zip_code
FROM   employee;
```

```
CREATE view emp_dept20
AS
SELECT first_name, last_name, middle_initial,
       street_address, state, zip_code
FROM   employee
WHERE  deptno = 20;
```

SQL: CREATE VIEW…

- CREATE VIEW is very similar to the 'CREATE TABLE AS' statement:
  - **CREATE VIEW** CSStudents **AS**
  - SELECT *
  - FROM Student
  - WHERE StdMajor = 'CS';
  - **WITH CHECK OPTION;**
- Once the view is created, it can be queried and (sometimes) updated the same as a base table
- Views are **dynamic** – changes to the base table(s) are reflected in the view

SQL: CREATE VIEW… WITH CHECK OPTION

The WITH CHECK OPTION clause ensures that updates to an updateable view don't violate any constraints on the underlying tables:

```
CREATE VIEW CSStudents AS
SELECT *
FROM Student
WHERE StdMajor = 'CS'
WITH CHECK OPTION;
```

This means that if StdMajor in STUDENT had been defined as a having a limited set of allowable values by a CHECK constraint, it would not be possible to violate this through updating the view

Uses of views

1. Provides a way to hide rows or columns (give the users access to the data that they need to do their job)
2. Can update base tables (sometimes) (insert,delete,update)
3. Complex SQL syntax can be hidden
4. Display outcomes of calculations
5. Able to isolate the user view of the data from the actual table data

3. Layer built-in functions
4. ~~Provide level of isolation between table data and users' view of the data~~
5. Assign different processing permissions to different views of the same table
6. Assign different triggers to different views of the same table

(There are examples of each of these uses in the text, p361-371)

View updateability

- Views are 'virtual' tables – unlike the base tables they are derived from, they do not contain any actual data
- However, in some cases the base table(s) can be updated through the view, using exactly the same SQL commands (insert, delete, update) as for tables – when this can occur the view is said to be **updateable**
- The rules for view updateability are complex and DBMS-dependent, but basically the update must provide an unambiguous result on the base table(s)

Updateable views (simply)

Updateable:

- Views for a single table without computed columns and any columns that require a value are present in the view

Not updateable:

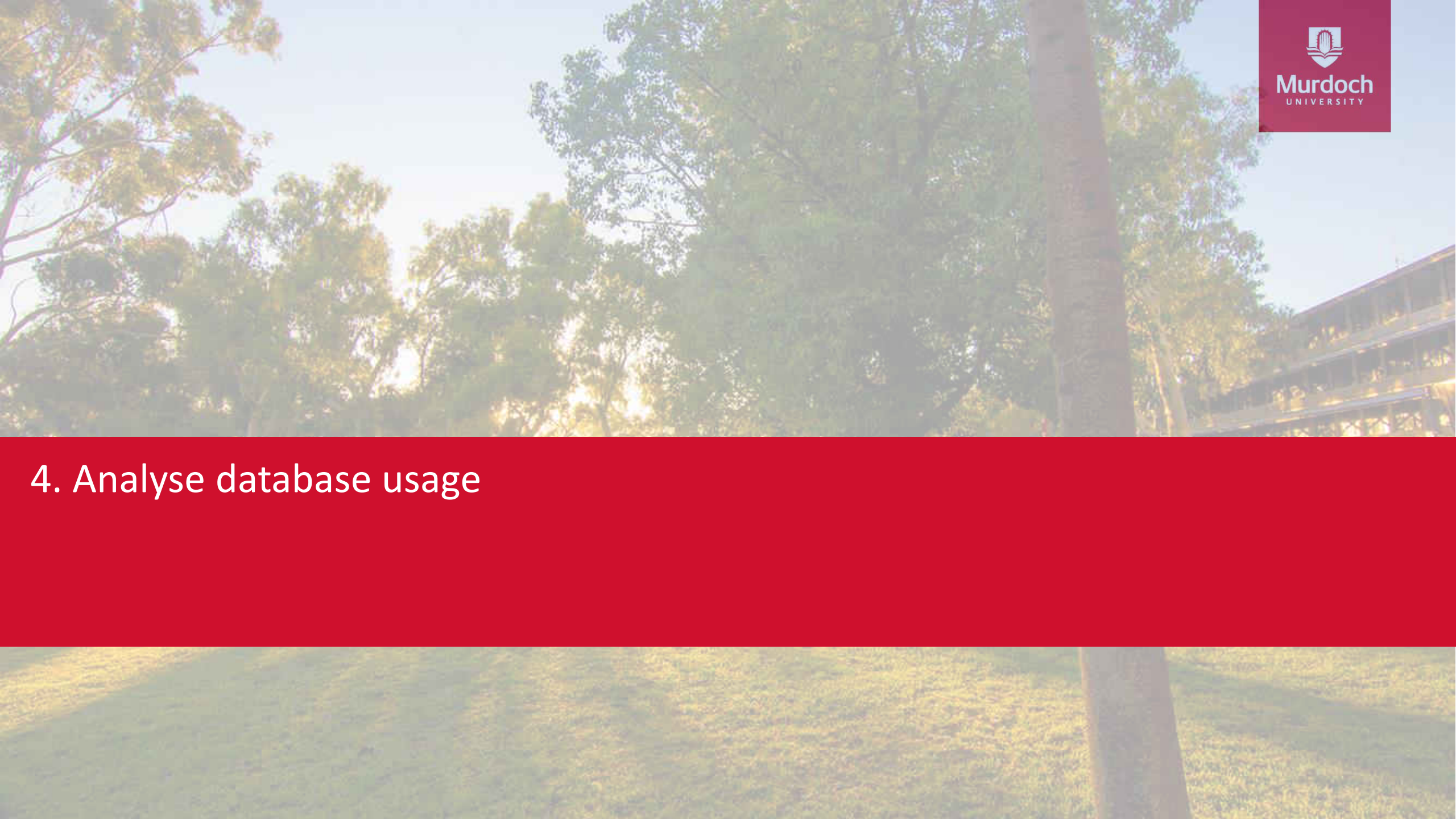
- View have an aggregated or computed column
- Views are created from some joins

Possibly updateable:

- A primary key on a single table with some required columns not having a value may be able to do update (if requirement column not involved) and delete but not allowed insert (cos don't know what to put in a required column in order to stop integrity constraint)
- Multiple tables, updates could be allowed on the most child table in the view only if rows of that table can be uniquely identified

The take-aways…

- Views are ‘virtual’ tables – a selection of row/columns based on the base tables for a variety of purposes
- Views permit simplification and abstraction of the underlying tables for the user
- Views are *dynamic* – updates to the base tables are reflected in the view
- However, the reverse does not apply: *the base tables can only be updated through the view in certain limited circumstances*



4. Analyse database usage

Transaction analysis (Topic 8)

- In physical design, we need to determine the optimal file organisations and the indexes that are required to achieve acceptable performance (Topic 8)
- Many of the choices in this phase are mutually exclusive
  - A file organisation good for bulk uploads will perform poorly for just about everything else!
- So we need to know how the database will be *used* – this is the first step

Transaction analysis

- An understanding of the transactions the database will run has a MAJOR impact on physical design (80/20 rule)
- Not all transactions will be known at design time, but the designers should have an understanding of the most important ones
- Analysis of the transactions will include criteria such as:
  - frequency
  - business (or system) impact
  - peak load
- Also need to know high-level functionality of the transactions, such as:
  - attributes that are frequently updated
  - search criteria used in a query

Transaction analysis - tools

There are many tools that can be used for analysing transactions, e.g.

- **Transaction/ Relation Cross Reference Matrix (CRUD)**
  - Shows Create (C), Read (R), Update (U), and Delete (D) data accesses to tables
- **Transaction Usage Map (TUM)**
  - Shows the pattern of data accesses and their frequency

These analyses can be presented in a transaction analysis report as part of the logical or physical design documentation

CRUD matrix

	Create Invoice	Accounts Payable	Inventory	Project Management	Upload to Data Warehous
Invoice	CRU	R	R	R	R
Project	R			U	R
Invoice Line	CRU	R	R	R	R
Part	R	R	U	R	R
Supplier	R	RU			R

Transaction Usage Map

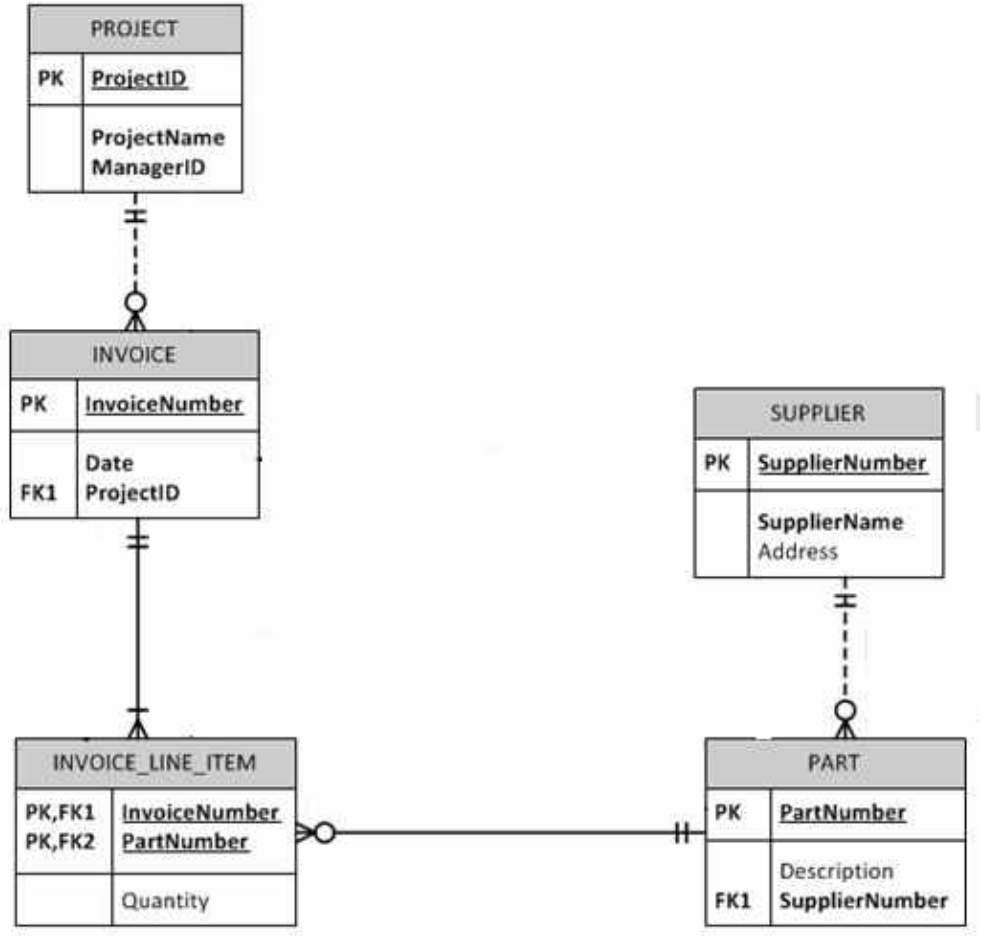
Identifies the major transactions and analyses the usage patterns and usage paths required for each transaction

Combines information for the transaction:

- Data volume
- Average
- Peak (volume and time)
- Expected growth
- Nature of participation of tables/columns

•Also known as: logical access maps, action diagrams

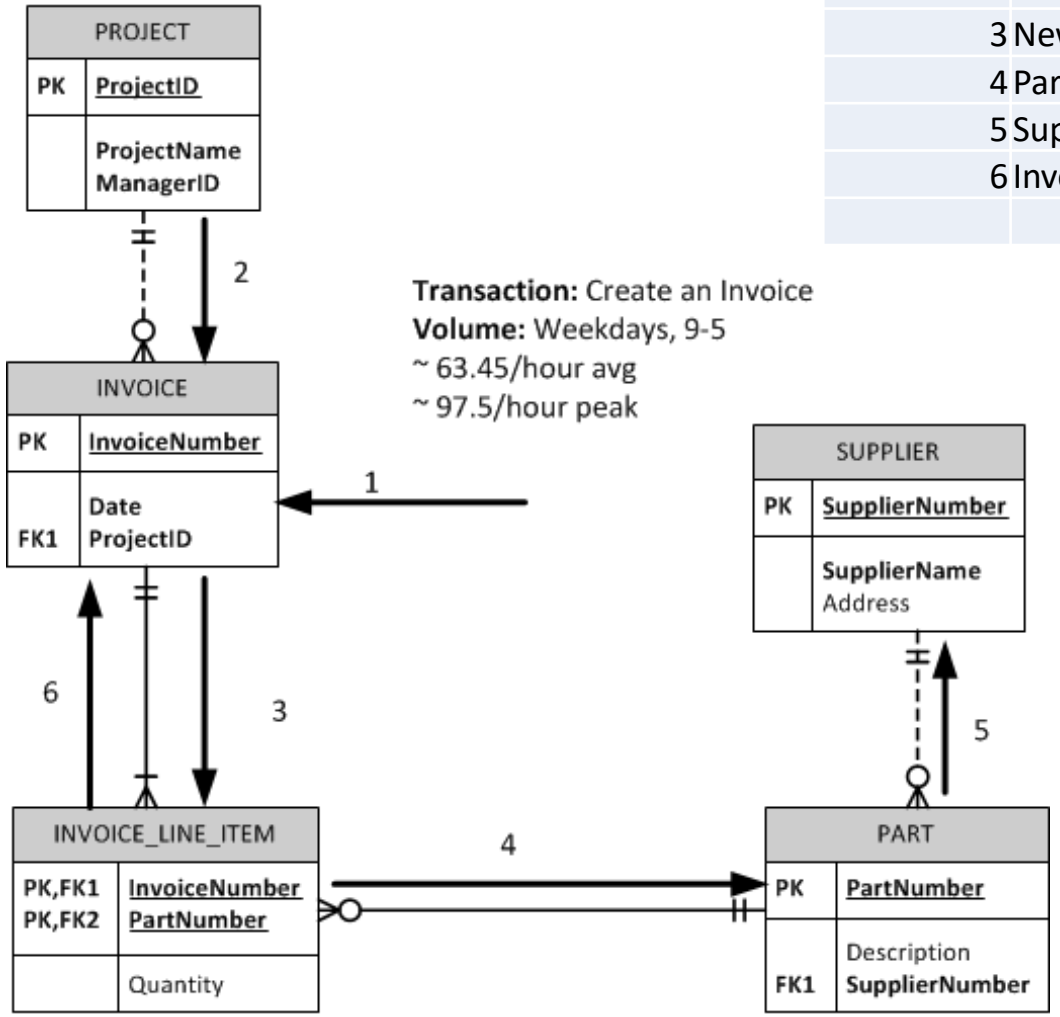
Transaction Usage Map
Consider a typical transaction: Updating an invoice for a project



- Involves:
- Create Invoice details
  - Create invoice
  - Check project exists
- Add Invoice Line details
  - Create invoice line
  - Check invoice exists
  - Check part exists
  - Check Supplier exists

TUM: Example

Step	Name	Access type	Refs per Trans.	Peak Refs/Hr
1	Create new invoice	C	1	97.5
2	Project	R	1	97.5
3	New Invoice Line	C	10-15	1462.5
4	Part	R	10-15	1462.5
5	Supplier	R	10-15	1462.5
6	Invoice	R	10-15	1462.5
	Total References:		42-62	4582.5



- We can add the number of expected transactions to the map, showing the number of times each table is accessed

Estimate disk space requirements

- ~~We need to know how much disk space the database will require~~
- ~~Can calculate from space required for populated base tables and other objects such indexes~~
- ~~Record length (= sum of field lengths) x estimated number of records~~
- ~~System tables will also require storage space~~
- ~~Must allow for expected growth~~

The take-aways…

- Before we can create an efficient physical design, we need to know how the database will be used
- **Transaction analysis** can be used to identify characteristics such as:
  - Areas of potential high load (average, peak)
  - Frequently queried tables/columns that may benefit from indexing
  - Tables that will grow rapidly
  - Candidates for possible denormalisation

5. Denormalisation

Denormalisation ("controlled redundancy")

- Additional choice in physical database design stage
- A database that is fully normalised to 3NF is consistent, is easier to update, requires less coding to enforce integrity constraints, is flexible and has minimal redundancy
  - However, it may not be the best in terms of *processing efficiency*

Denormalisation

- **Denormalisation** is about joining tables so query is more easier (better processing efficiency). So tables isn't often updated but queried a lot and performance is slow use denormalisation
- Advantages-
  - Denormalisation can provide improved performance by making query easier

- Disavantages-
  - Increased complexity for implementation to maintain consistency (because we are trading advantages we did normalisation in the first place)
  - Often "denormalisation" ends up with lower normal form, but may not (e.g. it may just include more nulls than the original tables)
  - Makes updates slower because retrieval is quicker (easier query)
  - Flexbility is reduced

- ~~Remember that denormalisation is a deliberate choice based on processing needs – it is not the same as not normalising in the first place!~~

When would denormalisation be appropriate?

- Denormalisation can provide better efficiency, but remember that:
  - It makes implementation more complex – to maintain consistency
  - It can sacrifice flexibility
  - It may speed up retrievals but slow down updates
- As a general principle, if performance is unsatisfactory and the tables involved are *updated infrequently* but *queried a lot*, it may be worth denormalising

Examples of denormalisation strategies

- Including repeating groups from a 1:N relationship in the parent table
- Combining 1:1 relationships
- Creating a single table from a generalisation hierarchy
- Duplicating non-key columns in a 1:N relationship to reduce joins
- Duplicating foreign key columns to reduce joins

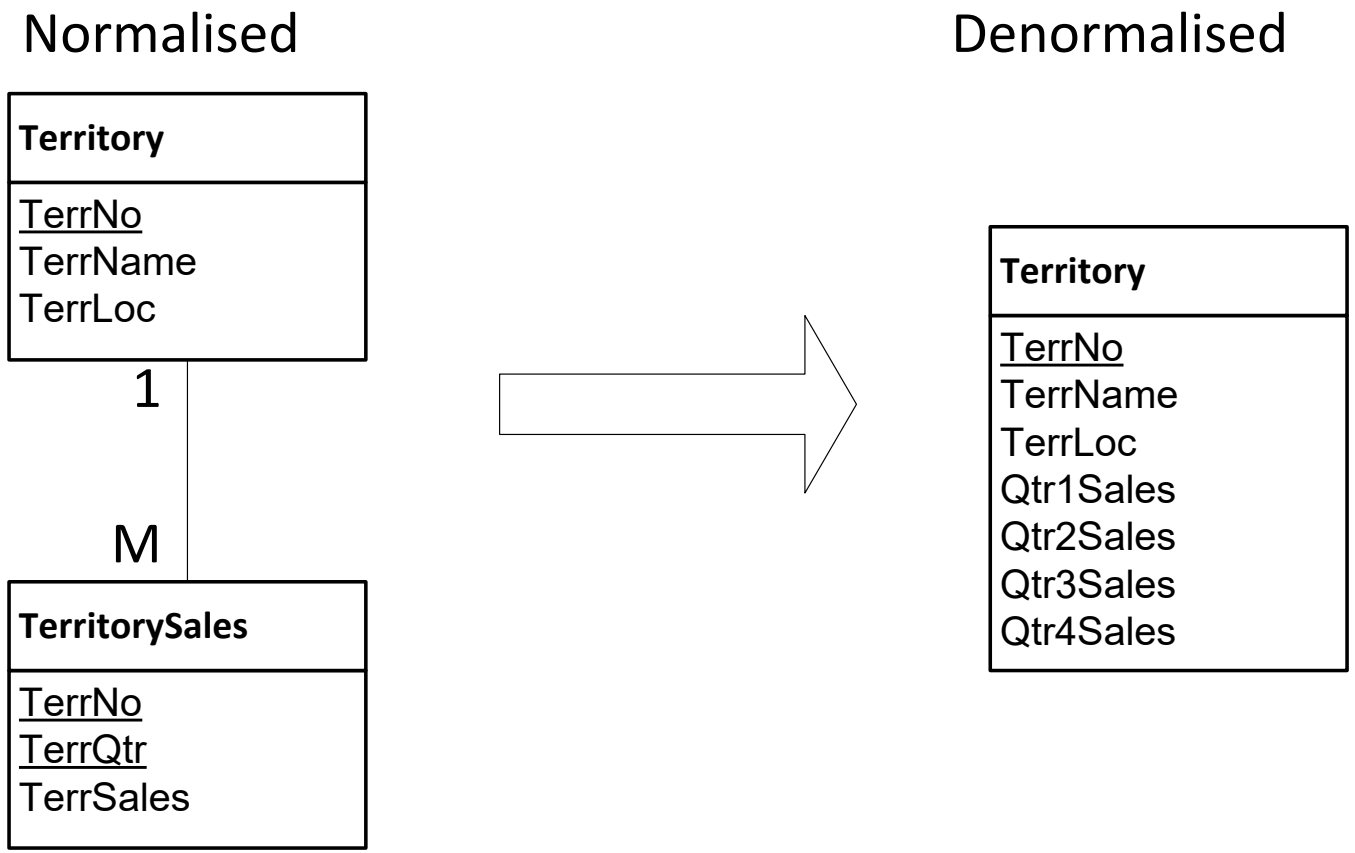
Denormalisation: Including repeating groups in the parent table

- ~~The rules of normalisation force repeating groups to be stored in an M-side table separate from an associated 1-side table~~
  - ~~e.g. Territory and quarterly TerritorySales~~

- Put repeating group as a value in the parent table rather than attribute (??). For repeating group accessed often with it's parent

- ~~If a repeating group is *always* accessed with its associated parent table, denormalisation may be appropriate~~
  - ~~Repeat quarterly TerritorySales within Territory record~~

Denormalising a repeating group



Why is the denormalised solution less flexible? Because if terrSales is monthly sales and not quartelySales then we need to update more attributes

Denormalisation: Combining a 1:1 relationship

- STAFF and NEXT-OF-KIN are in a 1:1 relationship, with each Staff member optionally having one Next of Kin
- Denormalise: Include (NEXT-OF-KIN) relation1 details in (STAFF) relation2

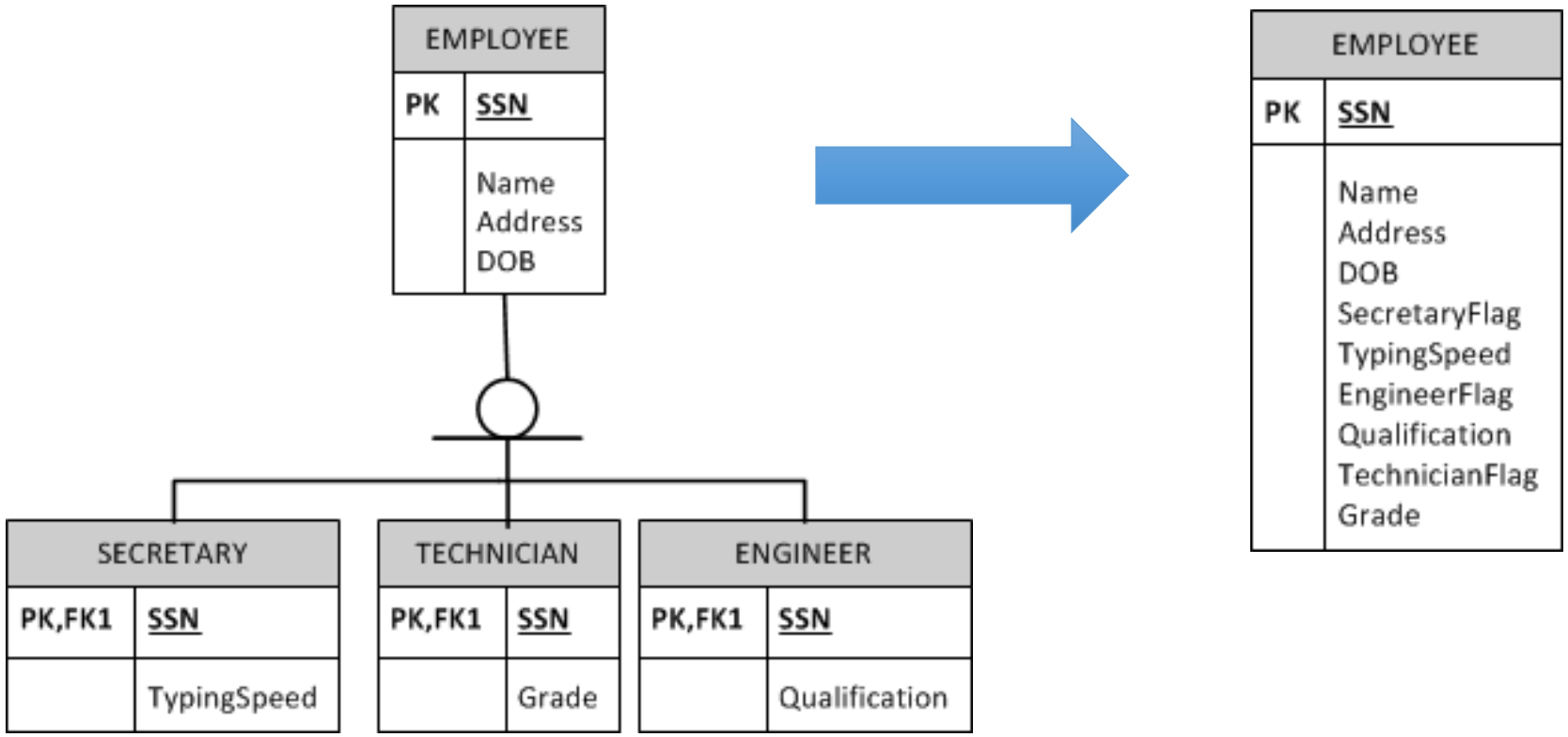
- Disadvantage-
  - New table has possibly many nulls which is wasted space
- ~~New table will potentially have many nulls (wasted space) if few staff have next of kin – balance against improved performance in retrievals~~

Denormalisation: Creating a single table from a generalisation hierarchy

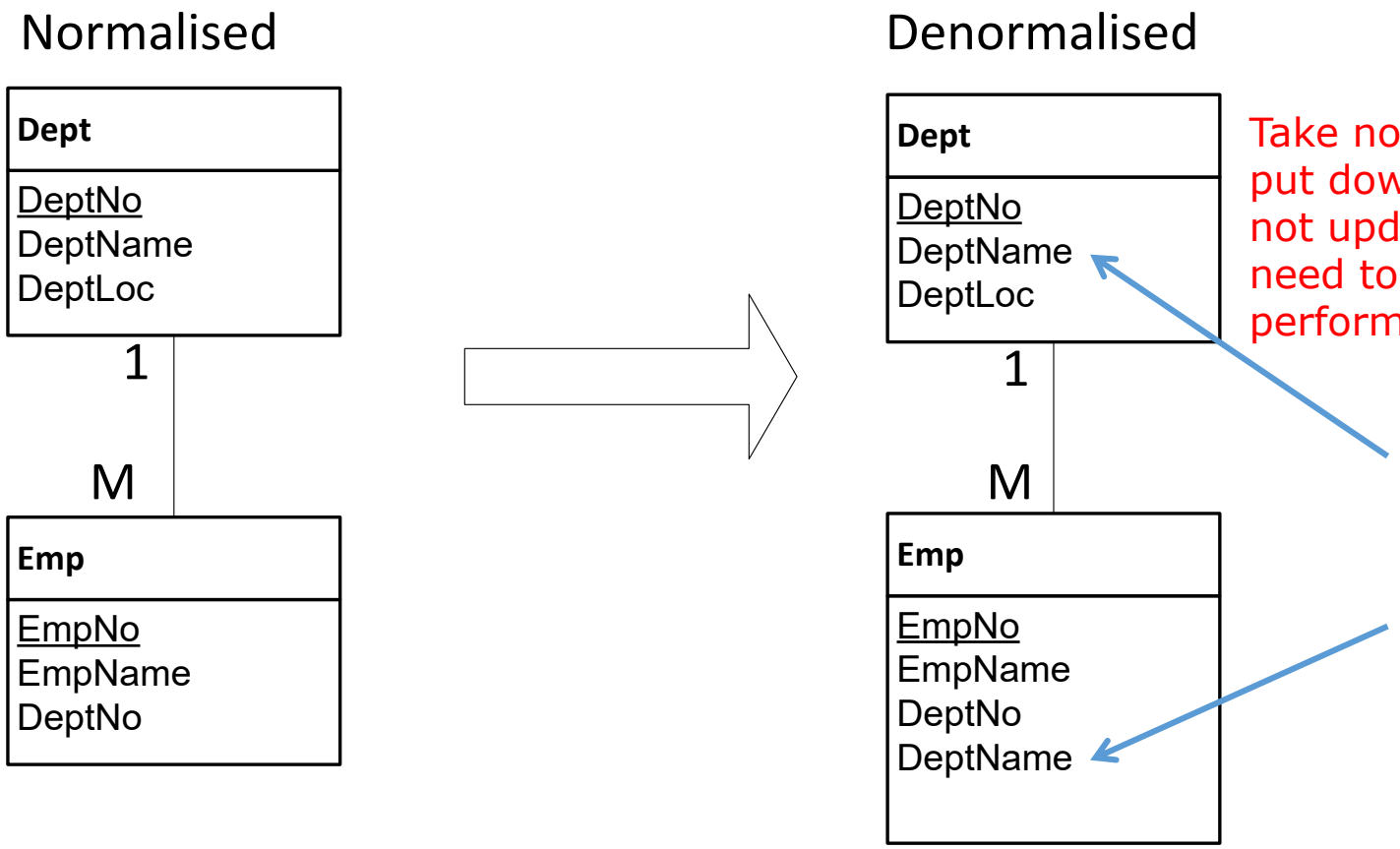
- Putting all the subtype and superTypes tables all into one table with all the attributes
- Disadvantage-
  - New table has possibly many nulls which is wasted space because attribute not applicable for the subtype will have a null

- ~~Again, need to strike a balance with the number of shared and specific attributes and the number of potential nulls~~

Denormalising a generalisation hierarchy



Denormalisation:
Duplicating non-key columns in a 1:N relationship to include meanings with codes)



What are the disadvantages of the denormalised solution?
Employee 2NF thus modification anomalies for 2NF exist

The take-aways…

- The careful introduction of controlled redundancy ("denormalisation") can improve performance in some situations
- These situations are often about reducing joins, which are one of the most expensive operations
- As a general principle, if performance is unsatisfactory and the table(s) *updated infrequently* but *queried a lot*, it may be worth denormalising
- The tradeoffs include larger tables, more nulls, and the increased potential for modification anomalies

6. Design security mechanisms

Database security

- The objective of database security is to only allow authorised users to do allowed activites at an authorised time EG
  - Students can enrol themselves in units in MyInfo
  - Academics can upload unit results
  - Advanced standing staff record credit and exemptions
- Must continuously maintain the design of the security throughout the life of the system(s) and database because requirements change
- ~~The processing rights and responsibilities for all users must be determined during the database design, and enforced using features of the DBMS and applications~~

Why do we need database security?

Data is an important organisation resource

- ~~It must be managed as any other valuable resource such as plant & equipment~~
- Corporate data is often stored electronically and the organisation relies on this data
- Data is needed to be kept both confidential and secure
  - e.g. client lists and details
- ~~Harm resulting to organisation from security breaches can be both tangible (loss of data) and intangible (e.g. reputation and client confidence)~~

Data is a shared resource and there are multiple users of the data and they have multiple sets of priorities

- Need to make sure those priorities don't conflict
  - e.g. retention requirements for different user purposes

Security (Reducing) risks

There are broad areas in which organisations should be seeking to reduce risk:

- Fraud as well as theft
  - Focus attention on reducing the opportunity for these to occur
- Privacy and confidentiality is lost
  - Refers to data which is crucial to the organisation, or, in the case of privacy, to the individual
- Integrity loss
  - Results in invalid or corrupt data
- Loss of availability
  - Increased requirement for 24/7 access to data

DBMS Security Guidelines 1

- **Run DBMS behind a firewall, but plan as though the firewall has been breached**
- **Apply the latest OS and DBMS service packs and fixes**
- **Use the least functionality possible**
  - Support the fewest network protocols possible
  - Delete unnecessary or unused system stored procedures
  - Disable default logins and guest users, if possible
  - Unless required, never allow all users to log on to the DBMS interactively
- **Protect the computer that runs the DBMS**
  - No user allowed to work at the computer that runs the DBMS
  - DBMS computer physically secured behind locked doors
  - Access to the room containing the DBMS computer should be recorded in a log

Copied from **Fig 9-16** in Kroenke, D.M., and Auer, D.J., 2016, Database Processing: Fundamentals, Design and Implementation, 14th Edition, Pearson.

DBMS Security Guidelines 2

- **Manage accounts and passwords**
  - Use a low privilege user account for the DBMS service
  - Protect database accounts with strong passwords
  - Monitor failed login attempts
  - Frequently check group and role memberships
  - Audit accounts with null passwords
  - Assign accounts the lowest privileges possible
  - Limit DBA account privileges
- **Planning**
  - Develop a security plan for preventing and detecting security problems
  - Create procedures for security emergencies and practice them

Copied from **Fig 9-16** in Kroenke, D.M., and Auer, D.J., 2016, Database Processing: Fundamentals, Design and Implementation, 14th Edition, Pearson.

Authentication, authorisation and access control

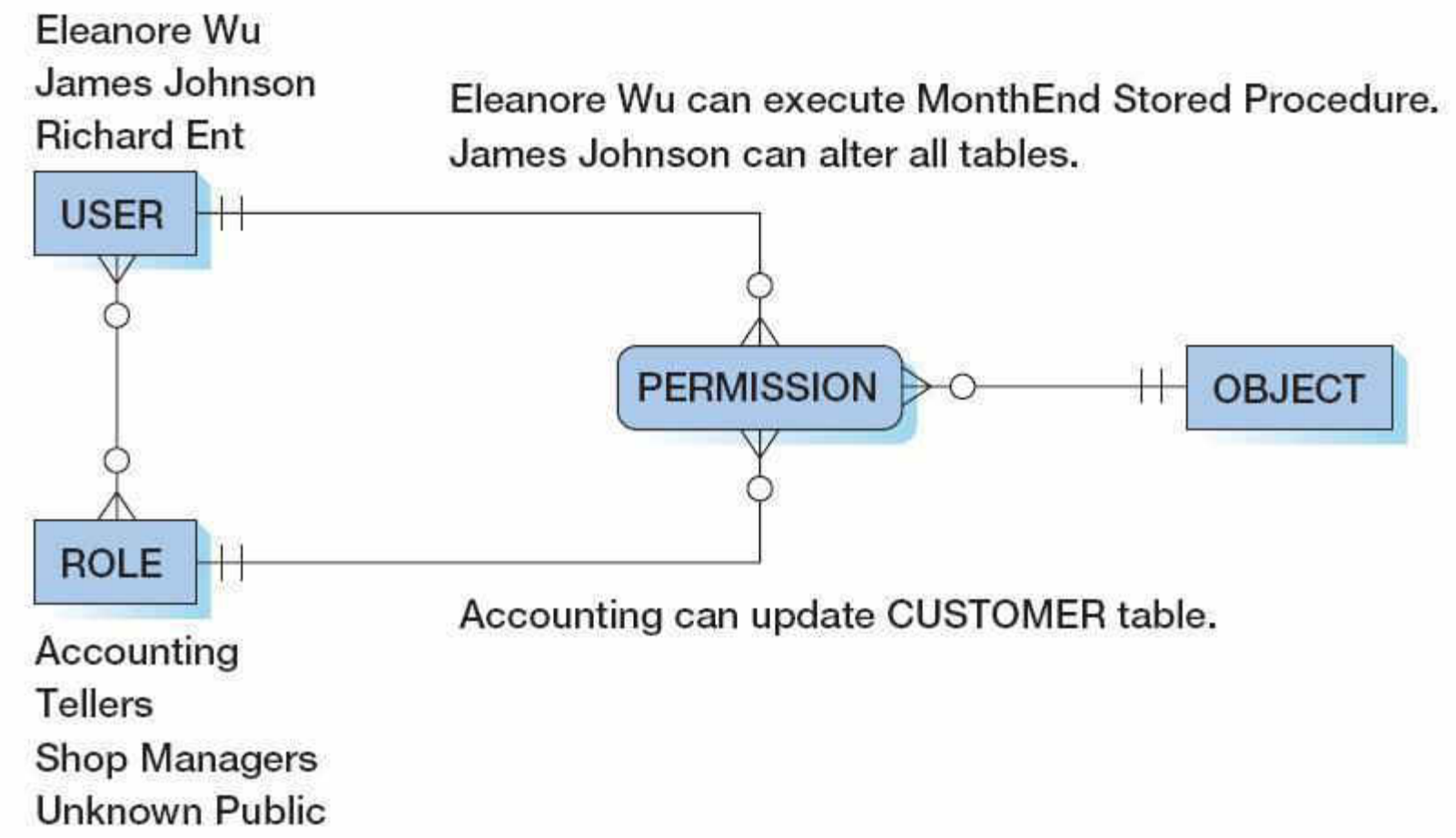
- Authentication
  - Checking whether the user is who they say they are
  - DBMS, OS authentication or a combination
- Authorisation
  - What a user is and isn’t allowed to do
  - Certain *Subjects* have access to certain *actions* on certain *objects*
- Access control

Access controls

Are about revoking and granting privileges

- A privilege enables a user to access or produce (read, write, or modify) some database item (such as a table, view, and index) or to execute certain DBMS tools
- Privileges are granted to users or roles to do the tasks needed for their jobs
- Granting as few privileges for a user or a role is preferred from a security perspective
  - i.e., only grant privileges when they are needed

A model of DBMS security



Copied from **Fig 9-15**in Kroenke, D.M., and Auer, D.J., 2016, Database Processing: Fundamentals, Design and Implementation, 14th Edition, Pearson.

DAC and MAC

- Discretionary Access Control (DAC)
  - ~~Most common approach in DBMSs~~
  - Access rights is worked out by owner
  - Typically *identity-based access control*: Owner specifies other users who have access
- Mandatory Access Control (MAC)
  - Access right is granted by rules
  - Also called *rule-based access control*

SQL GRANT

- SQL standard supports DAC through the GRANT and REVOKE commands
- The GRANT command gives privileges to users, and the REVOKE command takes away privileges.

  - **GRANT** {Privilege List | ALL PRIVILEGES}
  - **ON** Object Name
  - **TO** {Authorisation List|PUBLIC}
  - [**WITH GRANT OPTION**]

- If WITH GRANT OPTION is specified, the user can pass the privilege on to other users

GRANT: Table Privileges

The following privileges (among others*) can be granted on tables:

- **SELECT**
- **DELETE**
- INSERT [(ColumnName,….)]
- UPDATE [(ColumnName,….)]

GRANT **SELECT** ON Artist TO userA;

GRANT **UPDATE** ON Artist (LastName) TO userA;

*ALTER, REFERENCES, INDEX

GRANT…

Granting to PUBLIC

- Grants the privilege to any user who can log into the DBMS
- Should be exercised with caution!
  - *When might this be a good thing to do?*

```
GRANT SELECT ON Artist TO PUBLIC;
```

GRANT: Other Privileges

- Database Objects
  - User can create database structures such as databases, tables, triggers, etc
- System Privileges
  - User can execute certain system commands such as start and shutdown, start traces, manage storage
  - Usually reserved for DBA
- Program and Procedures
  - Users can execute programs or stored procedures

REVOKE

REVOKE [GRANT OPTION FOR]{PrivilegeList|ALL PRIVILEGES} **ON** ObjectName
FROM {AuthorisationList|PUBLIC}[RESTRICT|CASCADE]

- RESTRICT revokes the privilege only from the specified user
- CASCADE revokes it from any dependent privileges (those passed on by the grantees)
- GRANT OPTION FOR revokes the ability to pass on the privilege (revokes 'with grant option')

Roles and Groups

- Privileges can also be granted to roles and groups
- **Roles** are a set of privileges to objects that are consistent with the role
  - Users are then allocated to a role
  - Simplifies administration
    - STUDENT role
- **Groups**
  - Tend to be built in to particular DBMSs
    - SYSADM, DBADM, DBMAINT, SYSOP

Other DBMS-level security mechanisms

- **Views** can be used with GRANT to restrict access to parts of the database
- **Encryption** of important data in particular over the network
- Keeping track of database access through an audit trail can detect breaches to security
- **Regularly back up** the database and the log file

- Privileges can be managed on **stored procedures** so that access to the base data is via execution of the stored procedure, rather than via a view or the table itself

Application-level security

- When DBMS security methods are not enough extra security code can be outlined in the application
  - Application security in Internet applications is often provided on the Web server computer
- But use the DBMS level security methods first because
  - The closer the security enforcement is to the data, the less chance there is for infiltration
  - DBMS security features are faster, cheaper, and probably result in higher quality results than developing your own

The take-aways…

- An organisation's database is one of its most significant resources and must be protected, both during normal use and against deliberate attack and disasters
- The goal of database security is to ensure that only authorised users can perform authorised activities at authorised times
- Access controls are the most common DBMS security mechanisms
- The SQL GRANT and REVOKE statements are used by an object's owner to assign and remove privileges on objects to users and groups
- Additional security measures include views, backups and logs, encryption, auditing and integrity constraints, and through the application

7. Document the physical design

Document the physical design

- As always, all of the physical design decisions made during this phase must be documented – and the documentation kept up to date
- Much of the physical design will be implemented by the DBMS itself in the form of system tables in its internal data dictionary when the database is built
- Other documentation facilities are often provided by the DBMS environment or tools

Topic learning outcomes

After completing this topic you should be able to:

- Describe the activities in physical database design
- Design tables and integrity constraints for a target DBMS, based on the logical design
- Explain what views are and what they can be used for
- Draw a transaction usage map (TUM) for the most important transactions on a database
- Determine when denormalisation may be appropriate
- Use the SQL GRANT and REVOKE statements to manage user access to database objects
- Briefly describe other measures for DBMS-level security
- Document the physical database design

What's next?

In the next topic we look more closely at how the data in the database is physically held on secondary storage.

One of the responsibilities of the DBA is to determine the optimal file organisations and indexes that are required to achieve acceptable performance for the important transactions. These work together with the inbuilt query optimisation techniques of the DBMS to provide efficient processing.