Advanced Databases

Denormalization

Denormalization

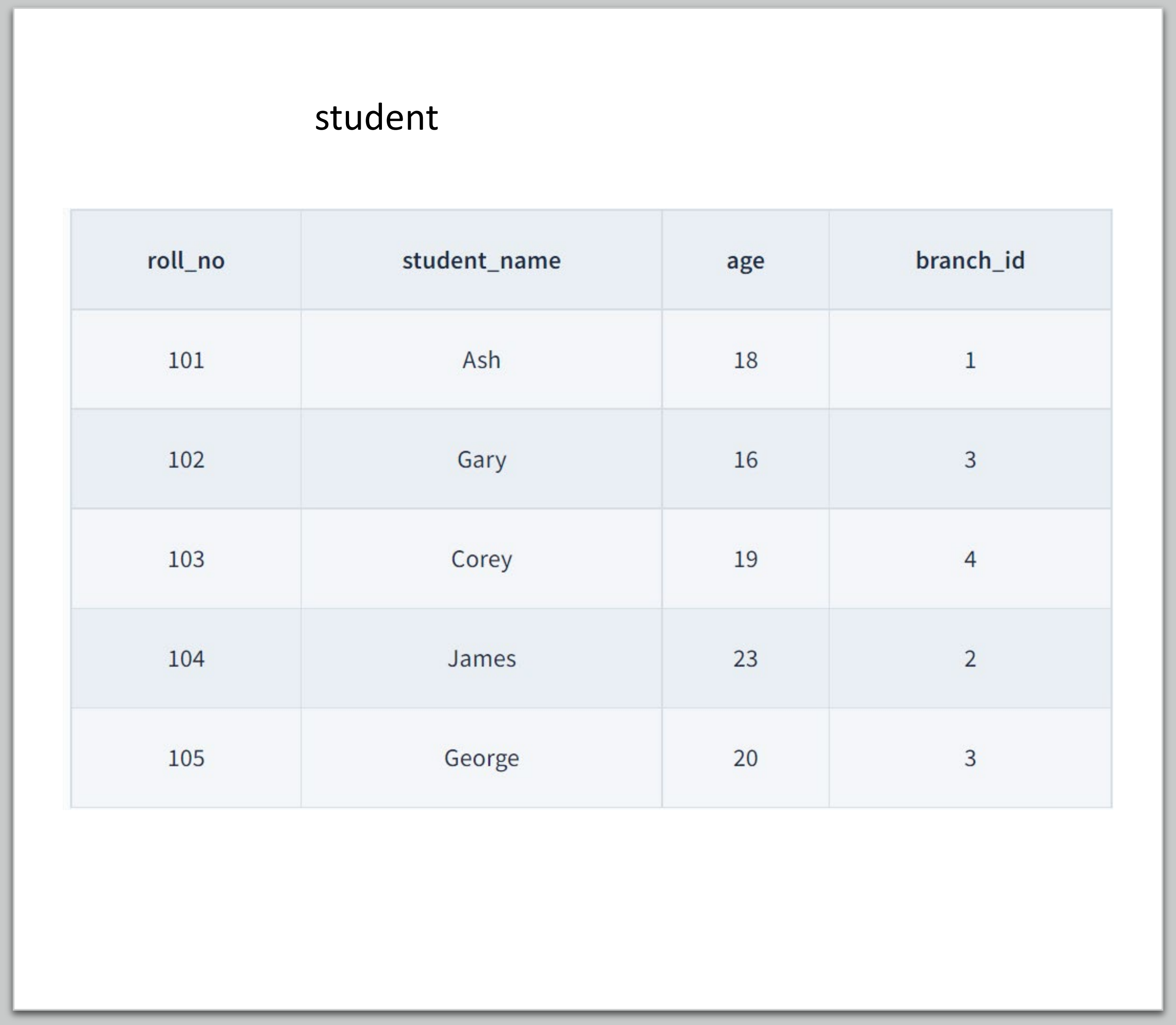
- Normalized (decomposed) tables require additional processing, reducing system speed.
- Normalization purity is often difficult to sustain in the modern database environment.
- The conflict between design efficiency, information requirements, and processing speed are often resolved through compromises that include **Denormalization**

Denormalization

- A database optimization technique
- Objective:
  - To optimize the efficiency of their database
- How:
  - Systematically add precomputed redundant data to a database to improve the read performance
- Why:
  - Can help avoid costly joins in a relational database made during normalization
- When is it done:
  - Can be done as part of design or delegated to the DBMS (handled by the DBA)

Example

- Three tables Student table, Branch table and HOD table after normalization.

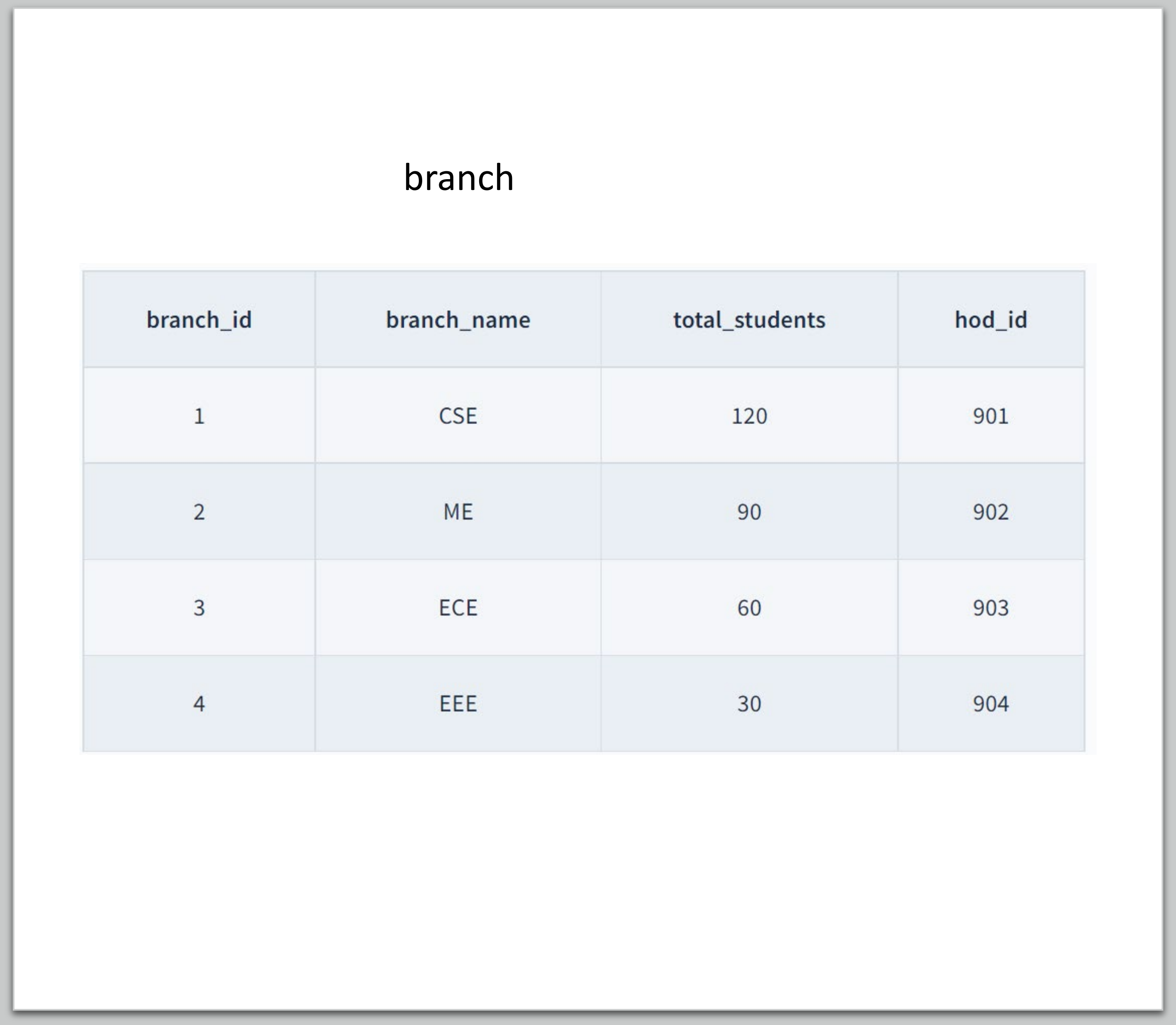


student

roll_no	student_name	age	branch_id
101	Ash	18	1
102	Gary	16	3
103	Corey	19	4
104	James	23	2
105	George	20	3

Example

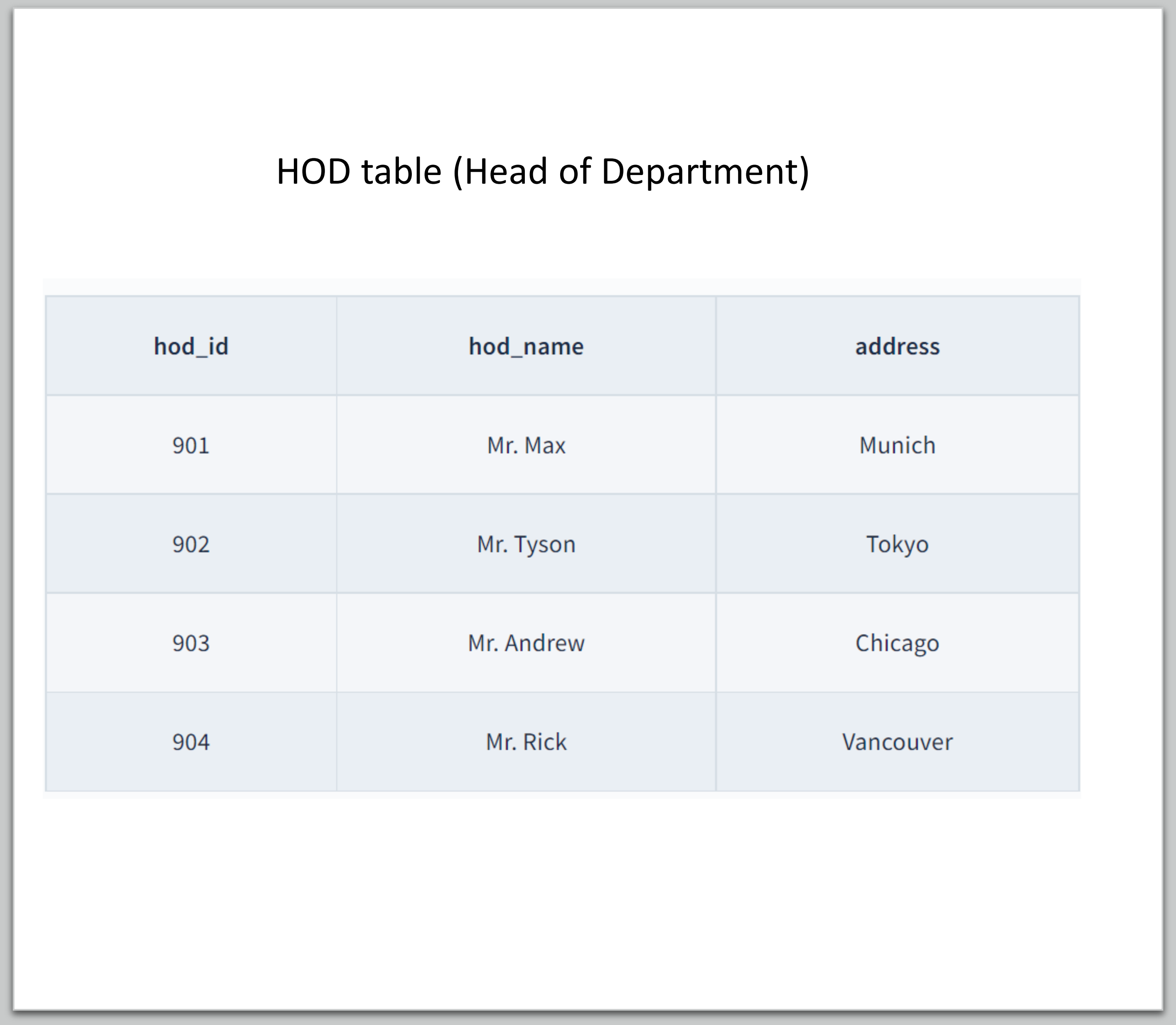
- Three tables Student table, Branch table and HOD table after normalization.


branch

branch_id	branch_name	total_students	hod_id
1	CSE	120	901
2	ME	90	902
3	ECE	60	903
4	EEE	30	904

Example

- Three tables Student table, Branch table and HOD table after normalization.



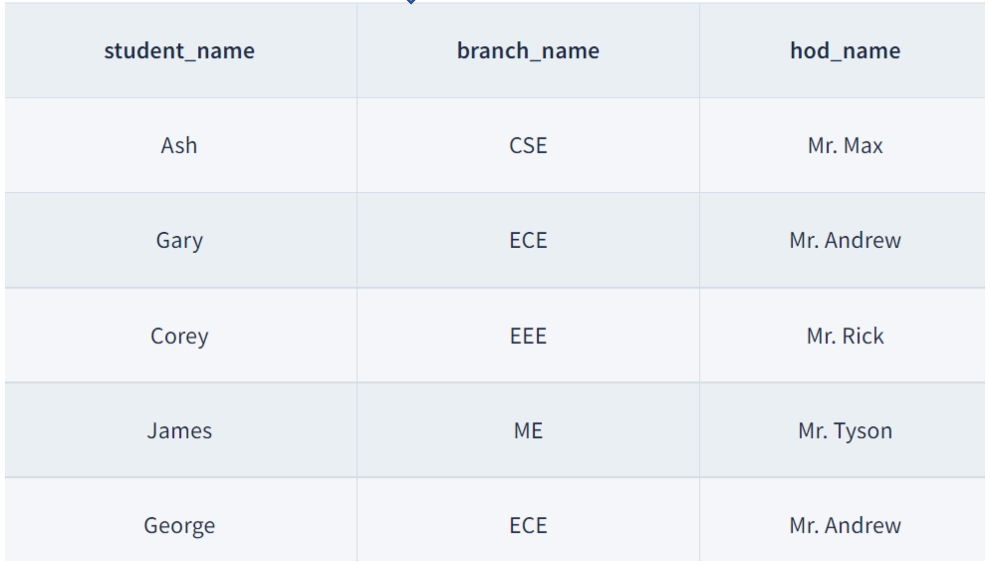
HOD table (Head of Department)

hod_id	hod_name	address
901	Mr. Max	Munich
902	Mr. Tyson	Tokyo
903	Mr. Andrew	Chicago
904	Mr. Rick	Vancouver

Example

- Suppose we want to retrieve all student names along with their branch name and hod name.
- We need to use a JOIN operation involving all three tables.
- If the amount of data held is small this is fine but in case for large amounts of data, joins on tables can take an excessively long time

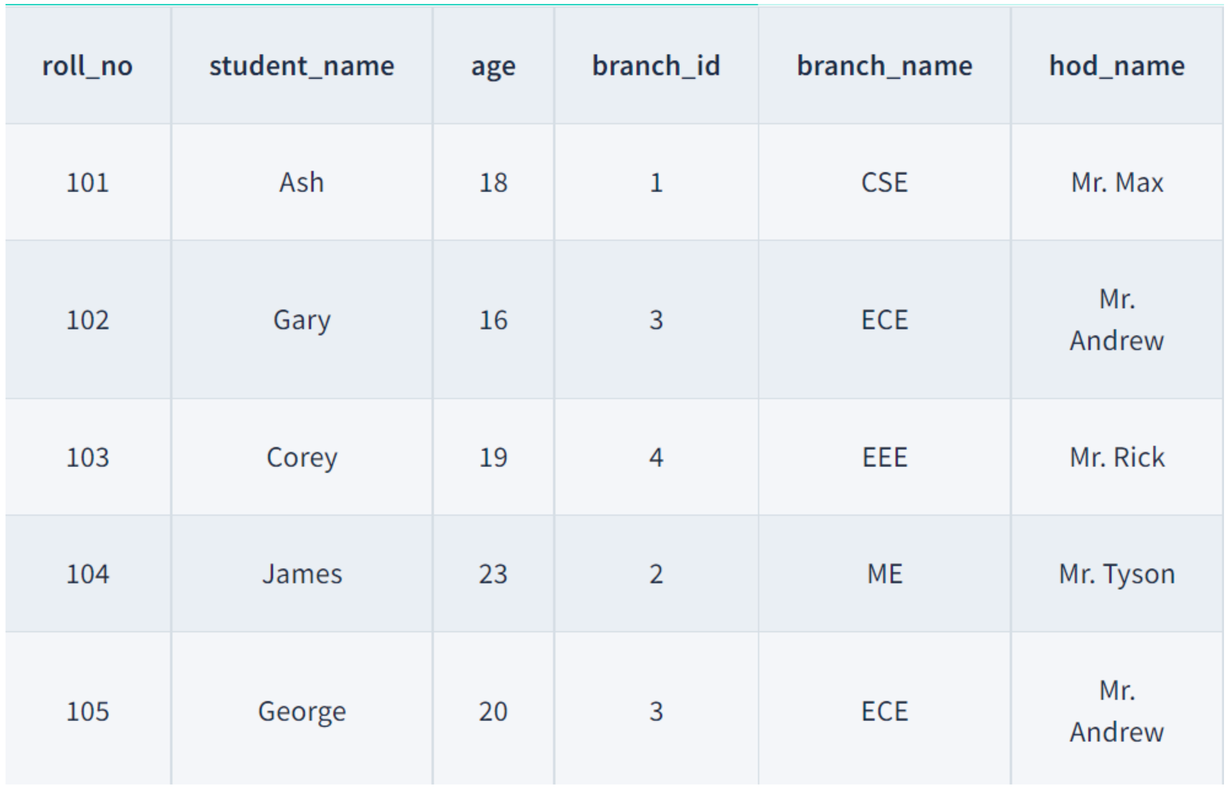
```
SELECT s.student_name, b.branch_name, h_hod_name
FROM student s
JOIN branch b
ON s.branch_id = b.branch_id
JOIN hod h
on b.hod_id = h.hod_id
```



student_name	branch_name	hod_name
Ash	CSE	Mr. Max
Gary	ECE	Mr. Andrew
Corey	EEE	Mr. Rick
James	ME	Mr. Tyson
George	ECE	Mr. Andrew

Example

- We can denormalize the database by including redundant data and extra effort to maximize the efficiency benefits of fewer joins.
- We can add the branch name, hod's name data from the Branch and HOD table respectively to the student table to optimize the database.
- Now we have a simple select on this table to handle our request
- Problems?



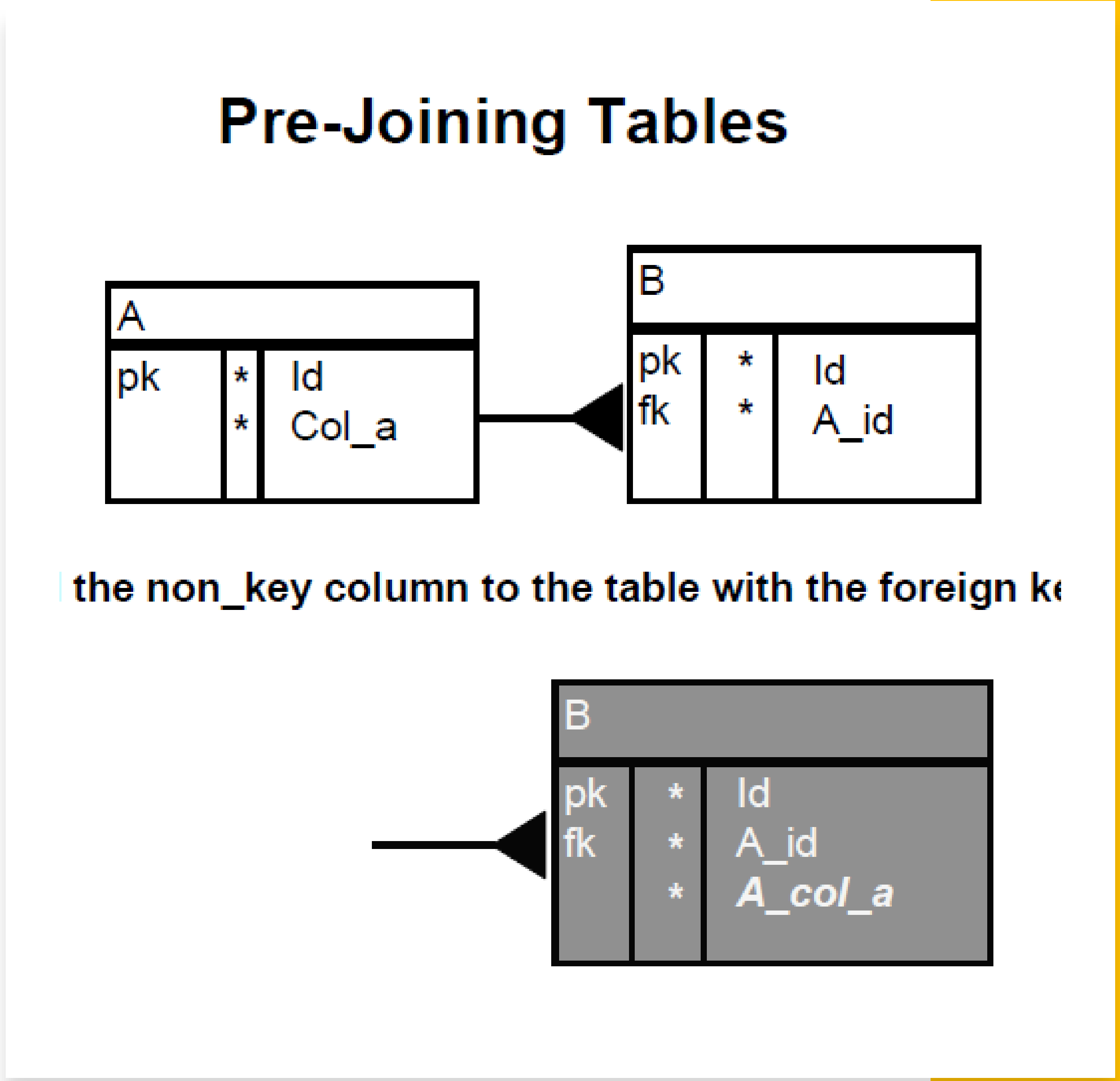
roll_no	student_name	age	branch_id	branch_name	hod_name
101	Ash	18	1	CSE	Mr. Max
102	Gary	16	3	ECE	Mr. Andrew
103	Corey	19	4	EEE	Mr. Rick
104	James	23	2	ME	Mr. Tyson
105	George	20	3	ECE	Mr. Andrew

Denormalization

Techniques

Pre-joining tables

- If you want to join tables where the actual value of the primary key, and consequentially the foreign key have no real business value, you can pre-join tables by including a non-key column in a table
- This will speed up specific queries.
- You must ensure that any application code updates the denormalized column each time the "master" column value changes in the referenced record.

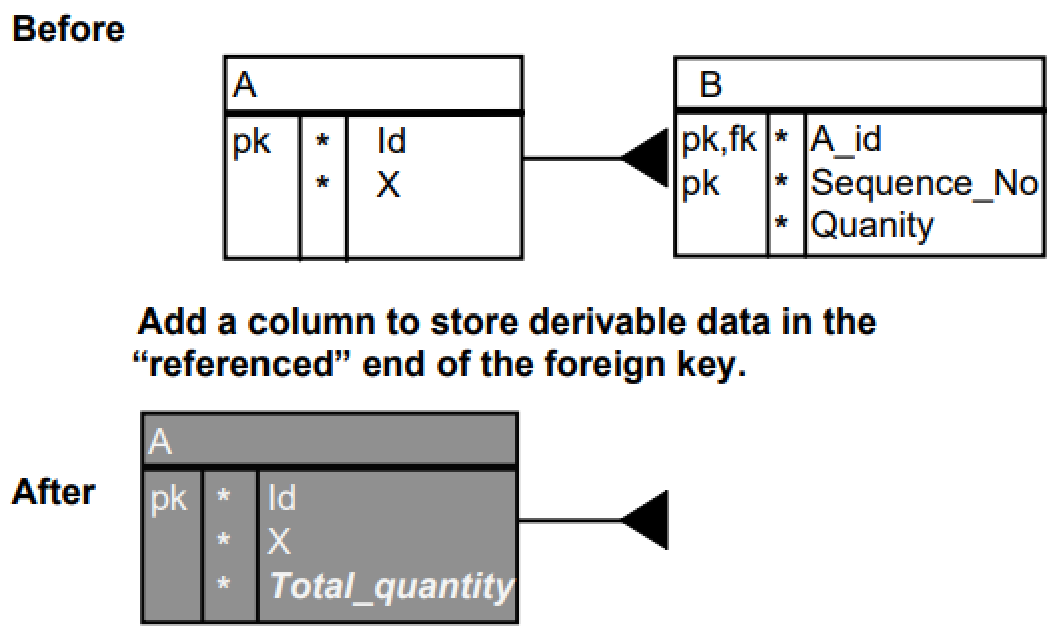


Pre-joining Tables

- When To use?
  - When frequent queries against many tables are required
  - When slightly stale data is acceptable (stale data = not 100% up to date)
- Advantages
  - Time-consuming joins can be avoided
  - Updates may be postponed when stale data is acceptable
- Disadvantages
  - Extra DML needed to update original non-denormalized column
  - Extra column and possibly larger indices require more working space and disk space

Storing Derivable Values

- If a frequently executed query, includes a calculation it can be worthwhile storing the results of the calculation.
- Where to store?
    - If the calculation involves detail records, then store the derived calculation in the master table.
    - Make sure to write application code to recalculate the value, each time that DML is executed against the detail records.
- Make sure that the denormalized derivable values cannot be directly updated.
    - They should always be recalculated by the system.

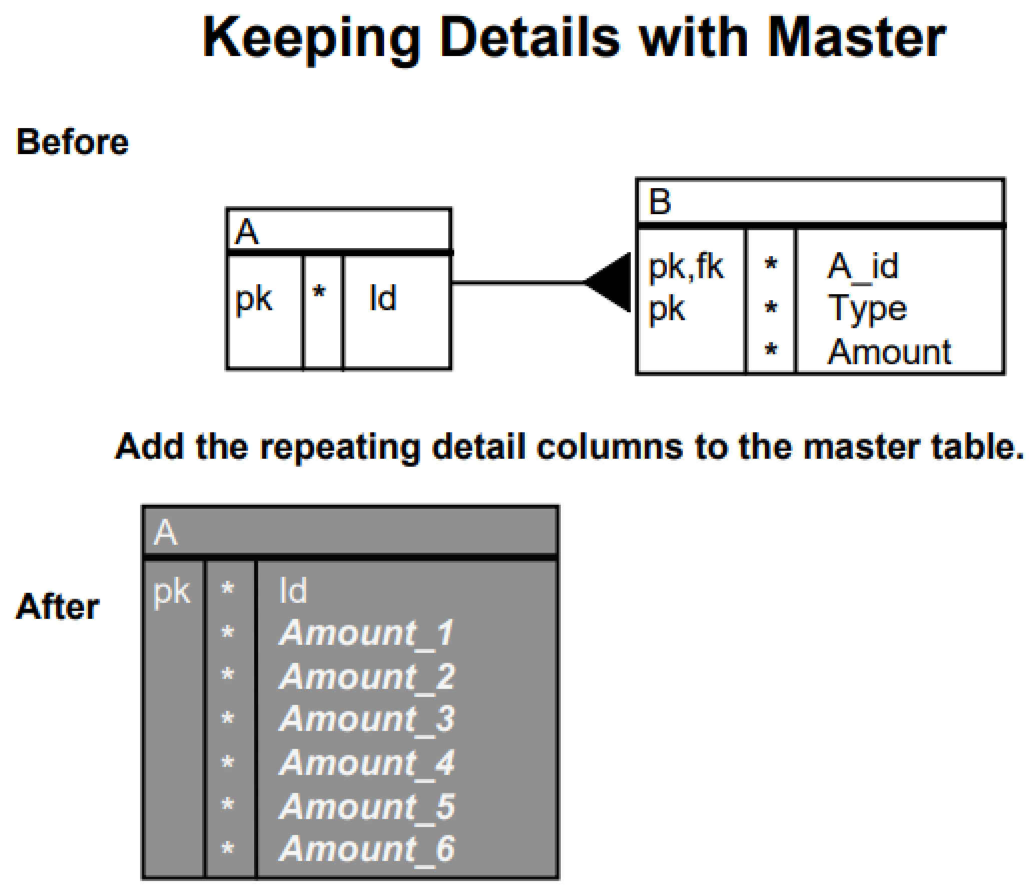

Before
A
pk
*
*
Id
X
B
pk,fk
pk
*
*
*
A_id
Sequence_No
Quanity
Add a column to store derivable data in the "referenced" end of the foreign key.
After
A
pk
*
*
*
Id
X
Total_quantity

Storing Derivable Values

- When To use?
  - Source values for the calculation are in multiple records or tables
  - The derivable values are frequently needed and the source values are not
  - The source values are infrequently changed
- Advantages:
  - Source values do not need to be looked up every time the derivable value is required
  - The calculation does not need to be performed during a query or report
- Disadvantages:
  - Any DML against the source data will require recalculation or adjustment of the derivable data
  - Data duplication introduces the possibility of data inconsistencies

Keeping details with master

- If the number of detail records per master is a fixed value (or has a fixed maximum) and usually all detail records are queried with the master, you may consider adding the detail columns to the master table.
- Works best when the number of records in the detail table are small.
- Will reduce the number of joins during queries.
- E.g.
  - A check in system where there is one record per person per day.
  - Could be replaced by one record per person per month, the table containing a column for each day of the month.

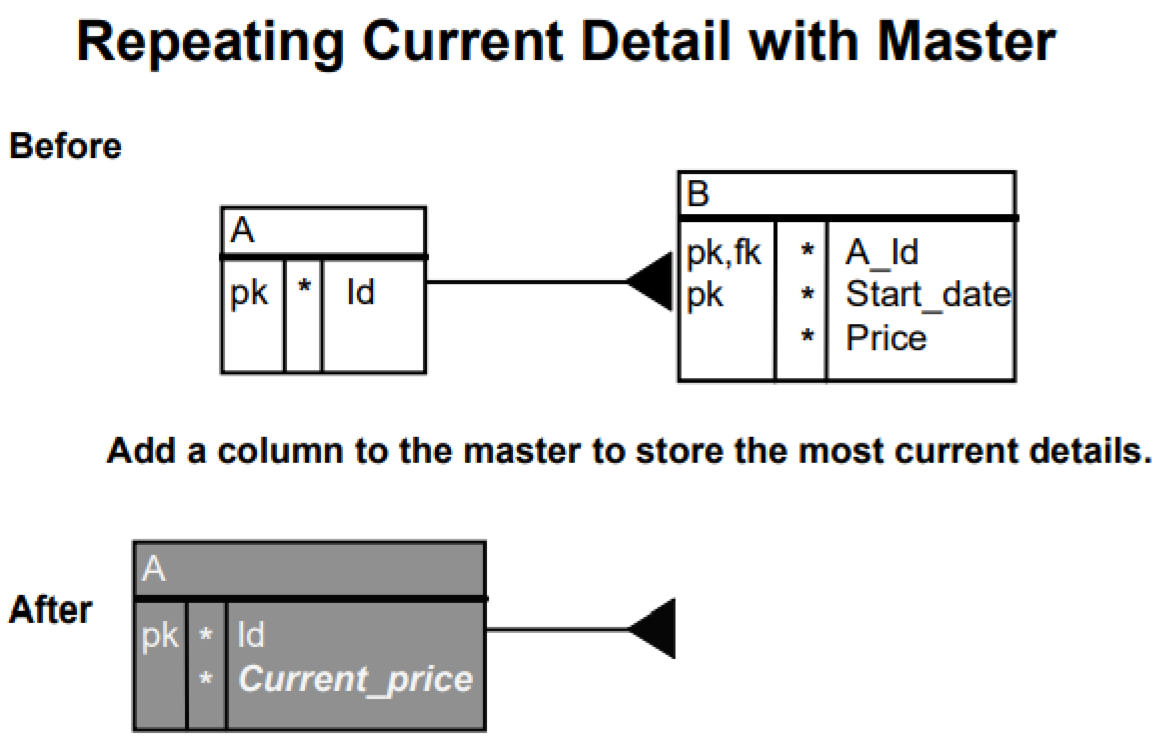


Keeping detail with master

- When To use?
  - When the number of detail records for all masters is fixed and static
  - When the number of detail records multiplied by the number of columns of the detail is small, say less than 30
- Advantages
  - No joins are required
  - Saves space, as keys are not propagated
- Disadvantages
  - Increases complexity of data manipulation language (DML) and SELECTs across detail values
  - Checks for detail columns must be repeated for all

Repeating single details with master

- When the storage of historical data is necessary, many queries require only the most current record.
- You can add a new foreign key column to store this single detail with its master.
- Make sure you add code to change the denormalized column any time a new record is added to the history table.

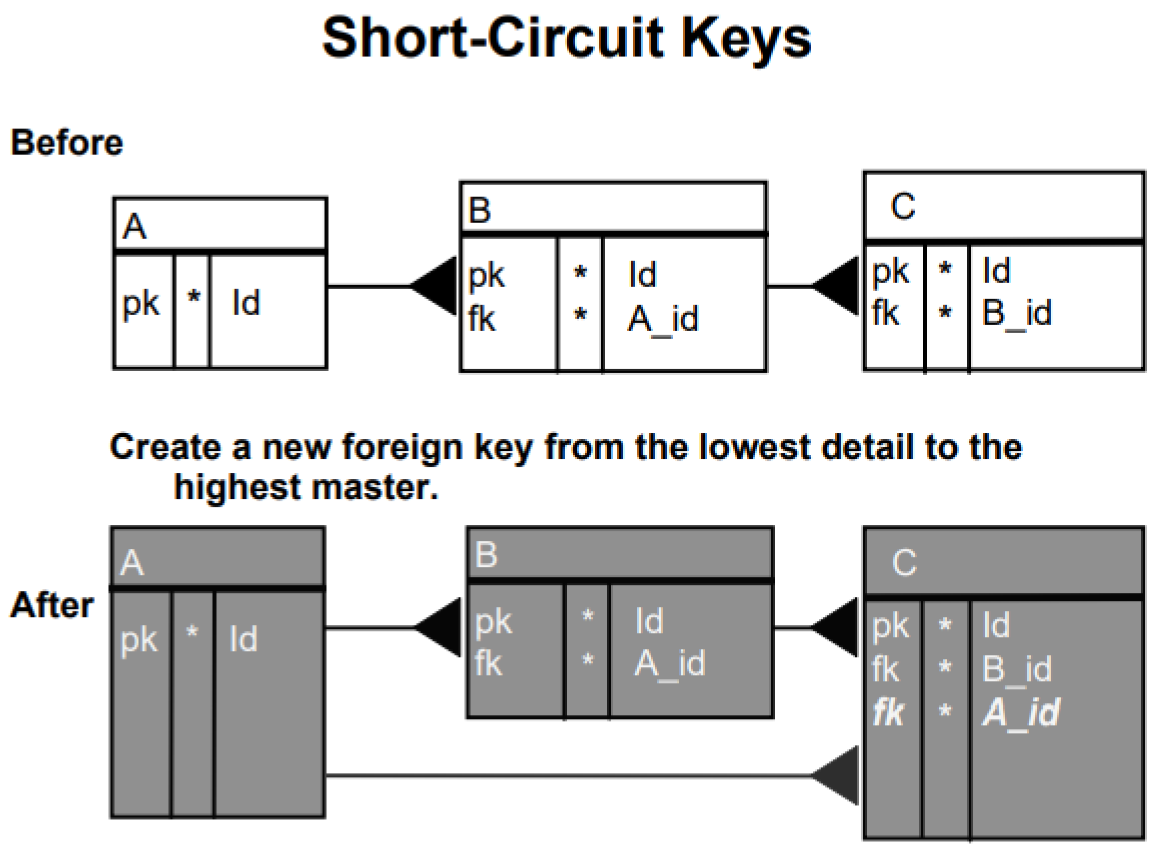

Repeating Current Detail with Master
Before
A
pk * Id
B
pk,fk * A_Id
pk * Start_date
* Price
Add a column to the master to store the most current details.
After
A
pk * Id
* Current_price

Repeating detail with master

- When To use?
  - When detail records per master have a property such that one record can be considered “current” and others “historical”
  - When queries frequently need this specific single detail, and only occasionally need the other details
  - When the Master often has only one single detail record
- Advantages
  - No join is required for queries that only need the specific single detail
- Disadvantages
  - Detail value must be repeated, with the possibility of data inconsistencies
  - Additional code must be written to maintain the duplicated single detail value at the master record.

Short Circuit Keys

- If your database design contains three (or more) levels of master detail, and there is a need to query the lowest and highest level records only, consider creating short-circuit keys.
- These new foreign key definitions directly link the lowest level detail records to higher level grandparent records.
- The result can produce fewer table joins when queries execute

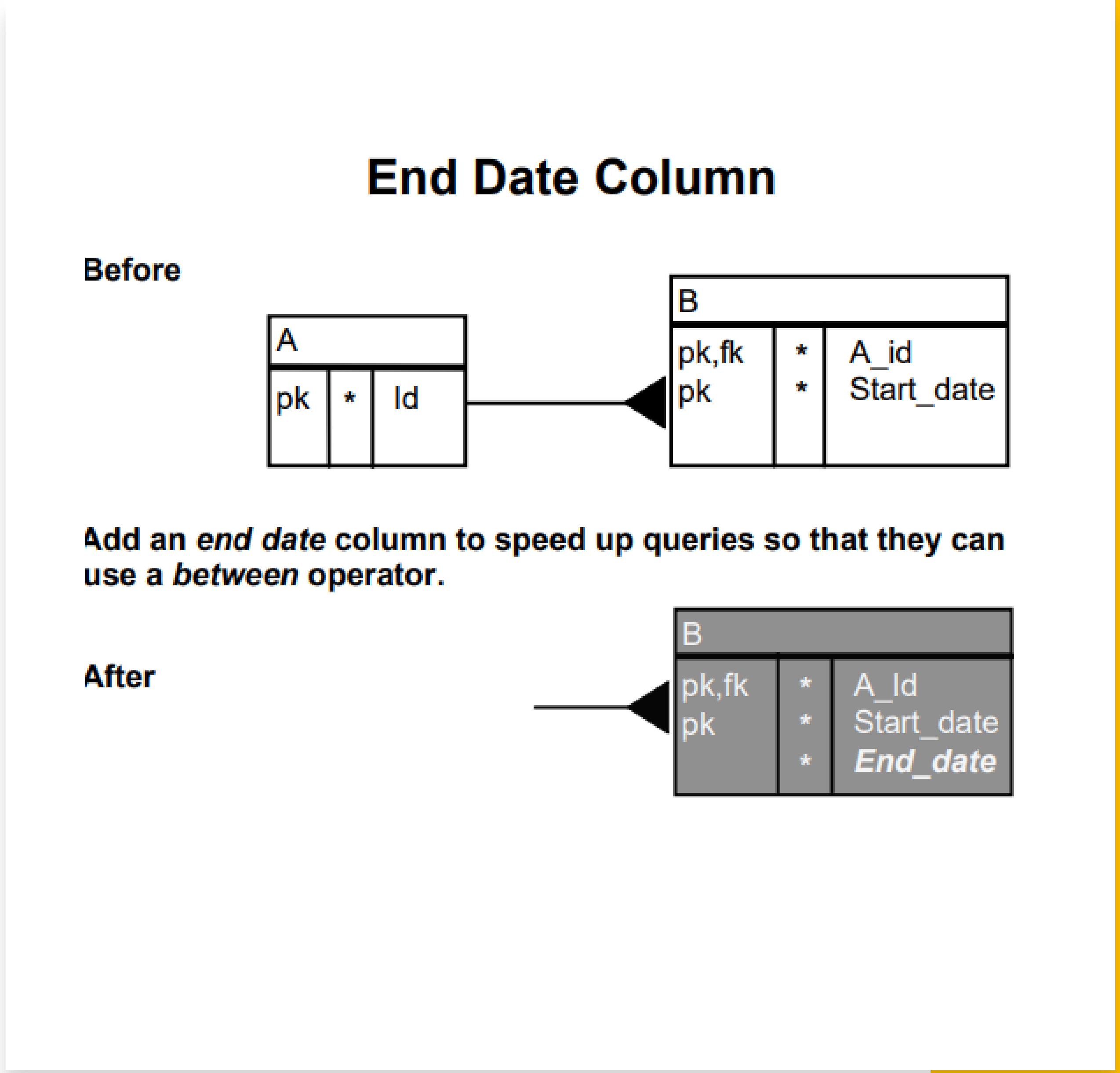


Short Circuit Keys

- When To use?
  - When queries frequently require values from a grandparent and grandchild, but not from the parent
- Advantages
  - Queries join fewer tables together
- Disadvantages
  - Extra foreign keys are required
  - Extra code is required to make sure that the value of the denormalized column (e.g. A_id) is consistent with the value you would find after a join with the master record (e.g. table B) .

End Date Column

- The most common denormalization decision is to store the end date for periods that are consecutive
- E.g. when the end date for a period can be derived from the start date of the previous period.
- If you do this, to find a detail record for a particular date you avoid the need to use a complex subquery.


End Date Column

Before

Add an *end date* column to speed up queries so that they can use a *between* operator.

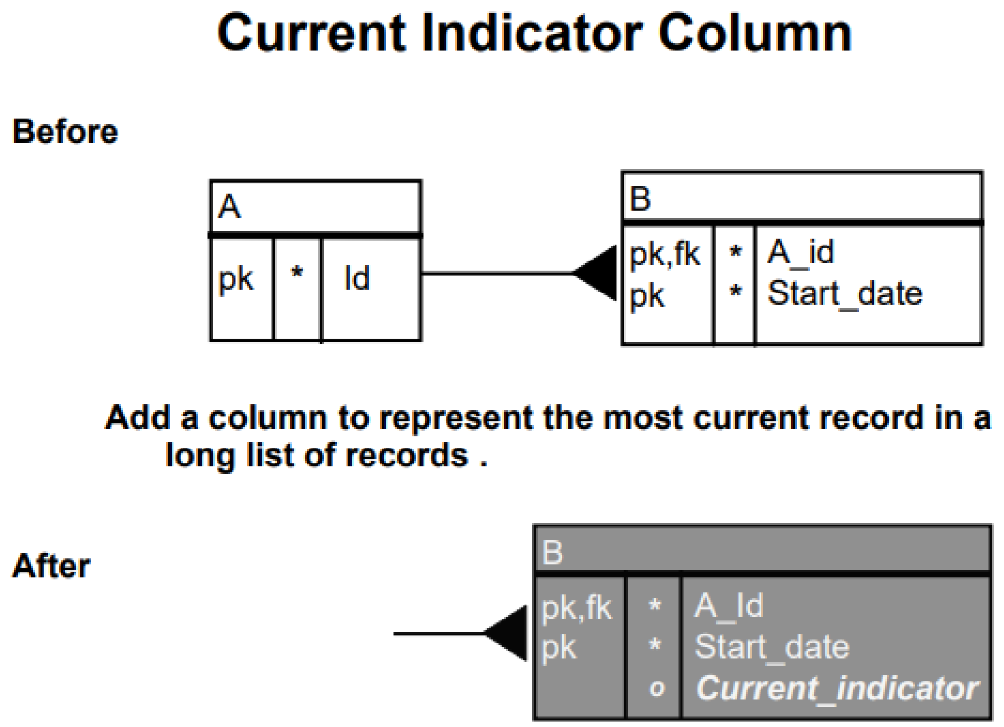
After

End Date Column

- When To use?
  - When queries are needed from tables with long lists or records that are historical and you are interested in the most current record
- Advantages
  - Can use the between operator for date selection queries instead of potentially time consuming synchronized subquery
- Disadvantages
  - Extra code needed to populate

Current Indicator Column

- Can be used in similar situations to the end date column technique.
- It can even be used in addition to an end date.
- Suppose most of the queries are to find the most current detail record.
  - With this type of requirement, you could consider adding a new column to the details table to represent the currently active record.
  - You would need to add code to update that column each time you insert a new record



Current Indicator Column

- When To use?
  - When the situation requires retrieving the most current record from a long list
- Advantages
  - Less complicated queries or subqueries
- Disadvantages
  - Extra column and application code to maintain it
  - The concept of “current” makes it impossible to make data adjustments ahead of time

Denormalizing Guidelines

Always create a conceptual data model that is completely normalized.

Consider denormalization as the last option to boost performance.

Never presume denormalization will be required.

To meet performance objectives, denormalization should be done during the database design.

Once performance objectives have been met, do not implement any further denormalization.

Fully document all denormalization, stating what was done to the tables, what application code was added to compensate for the denormalization, and the reasons

Plan for Continuous Assessment

Each week starting in week 3

You will work on a set of exercises

You will submit your work related to selected elements of these exercises/an additional exercise via Brightspace

Each week you will be told exactly what needs to be submitted for that particular exercise

Submission

There are two deadlines for submission:

- Week 8 (Provisional Date) – W/Ending Friday 11th November 2022 (20%) – Submit all exercises covered up to this point
- Week 13 (Provisional Date) – W/Ending Friday 16th December 2022 (40%) – Submit all exercises covered from week 8 to Week 13 (potentially resubmitting some previous exercises)