Triggers and Indexes

Databases

Ira Assent

ira@cs.au.dk

Data-Intensive Systems group, Department of Computer Science, Aarhus University, DK

Intended learning outcomes

- Be able to
  - Use triggers to define complex constraints
  - Use views as named queries
  - Discuss core principles in major indexing approaches to speed up database access

ira@cs.au.dk

Recap: More NFs

the new normal

- Describe goodness of schema with increasingly strict criteria
  - After 3NF, we have BCNF
    - Dependency preserving decomposition not always possible
      - Choose between less strict 3NF and BCNF that does not preserve dependencies
  - Multivalued dependencies (MVDs) capture new forms of dependencies
    - Attribute gives rise to set of values in another attribute
      - Course →→ Teacher
      - For any course there is a set of teacher values
  - Removing MVDs gives rise to decomposition into 4NF

For every nontrivial FD X → A, X superkey or A prime attribute — 3NF

For every nontrivial FD X → A, X superkey — BCNF

Trivial MVD X →→ Y in R:
1) Y subset of X (like trivial FD)
2) X ∪ Y = R

For every nontrivial MVD X →→ A, X superkey — 4NF

course	TA	teacher
dDB	aas	ira
dID	aas	ira
dDB	aas	schester
dDB	fra	ira
dID	zoffe	ira
dDB	zoffe	ira
dDB	fra	schester
dDB	zoffe	schester

Do not need to normalize to the highest possible normal form, typically used 3NF, BCNF, at most 4NF

- Denormalization: careful introduction of redundancy
- To speed up access with fewer joins, take workload into account

ira@cs.au.dk

Triggers

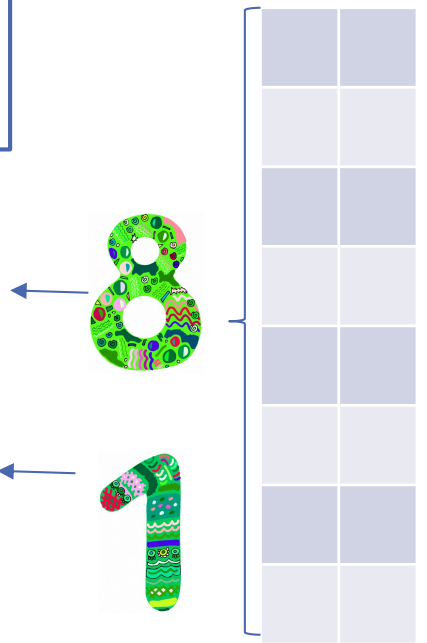
- Constraints have limited, fixed reactions to violations
- Triggers enable general reactions
  - Three components
    - event: AFTER, BEFORE
      for each of
      INSERT, DELETE, UPDATE
    - condition: any SQL Boolean expression
    - action: any sequence of SQL modifications
  - Also called **event-condition-action rules**
- Attribute and row checks are efficient, but not really expressive
- Triggers are much more expressive, and can often be efficient

https://dev.mysql.com/doc/refman/8.0/en/create-trigger.html
https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html

Row-Level Or Statement-Level

```
CREATE TRIGGER LogInsert
  AFTER INSERT ON People
  FOR EACH STATEMENT
  INSERT INTO LogFile
  VALUES('People', CURRENT_TIME);
```

- FOR EACH ROW
  - performs the action once for each row (=tuple)
- FOR EACH STATEMENT (not in MySQL)
  - performs the action once



ira@cs.au.dk

Referencing Old And New

- An `INSERT` implies there exists a variable for
  - a new row (if `FOR EACH ROW`)
  - a new table (if `FOR EACH STATEMENT`)
- A `DELETE` implies
  - an old row (if `FOR EACH ROW`)
  - an old table (if `FOR EACH STATEMENT`)
- An `UPDATE` implies both new and old versions

- These variables are referred to as
  - `NEW`, `OLD`
  - `ROW`, `TABLE`

ira@cs.au.dk

Example

```
CREATE TRIGGER PromotionOfficeDefault
  AFTER UPDATE ON People
  REFERENCING OLD ROW AS OldGuy
              NEW ROW AS NewGuy
  FOR EACH ROW
  WHEN (OldGuy.group ='phd' AND
        NewGuy.group ='vip' AND
        NewGuy.office IS NULL)
  UPDATE People
  SET office = 'Nygaard-355'
  WHERE userid = NewGuy.userid;
```

In MySQL

```
DELIMITER $$
CREATE TRIGGER
PromotionOfficeDefault
  AFTER UPDATE ON People
  FOR EACH ROW
  BEGIN
    IF (OLD.group = 'phd' AND
      NEW.group = 'vip' AND
      NEW.office IS NULL)
    THEN
      UPDATE People
      SET office = 'Nygaard-355'
      WHERE userid = NEW.userid;
    END IF;
  END$$
DELIMITER ;
```


ira@cs.au.dk

Choosing triggers

Using Sells(shop, product, price) and RipoffShops(shop),
maintain a list of shops that raise the price of any product by more than $1

```
CREATE TRIGGER MaintainRipoffs
  AFTER UPDATE OF price ON Sells
  REFERENCING
       ???
  FOR EACH ROW
  WHEN(B.price > A.price + 1.00)
  INSERT INTO RipoffShops
      VALUES(B.shop);
```

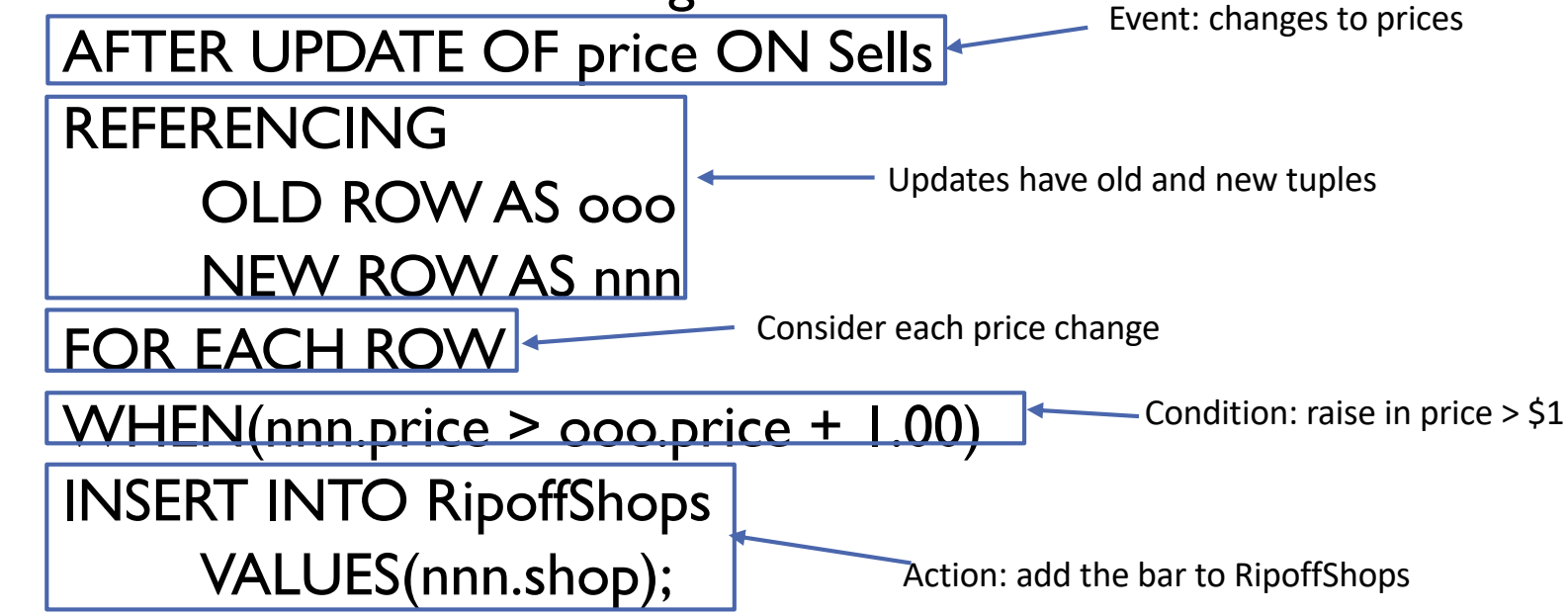
??? should be:

1. OLD TABLE AS A,
   NEW TABLE AS B
2. NEW TABLE AS A,
   OLD TABLE AS B
3. OLD ROW AS A,
   NEW ROW AS B
4. NEW ROW AS A,
   OLD ROW AS B

ira@cs.au.dk

Triggers put to work

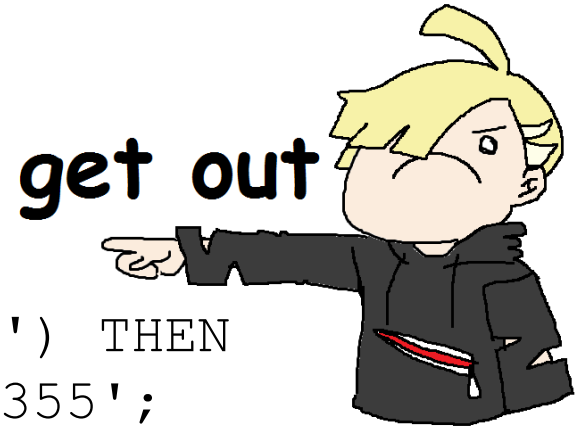
Using Sells(shop, product, price) and RipoffShops(shop), maintain a list of shops that raise the price of any product by more than $1.



ira@cs.au.dk

Example: BEFORE INSERT

```
CREATE TRIGGER GetOutOfMyOffice
    BEFORE INSERT ON People
    FOR EACH ROW
    BEGIN
        IF (NEW.office = 'Nygaard-357') THEN
            SET NEW.office = 'Nygaard-355';
        END IF;
```



- Access the new value of office in the insert and change it to a new value instead
  - No-one can be assigned my office Nygaard-357, but will be placed in Nygaard-355 instead

ira@cs.au.dk

Example: BEFORE UPDATE

```
CREATE TRIGGER StayOutOfMyOffice
    BEFORE UPDATE ON People
    FOR EACH ROW
    BEGIN
        IF (NEW.office = 'Nygaard-357') THEN
            SET NEW.office = 'Nygaard-355';
        END IF;
```

- Access the new value of office in update and change it to a new value instead
  - With these two triggers capture both inserts and updates
    - Similarly, if some office should be attempted to be updated to my office Nygaard-357, move to Nygaard-355 instead

ira@cs.au.dk

Example: complex condition

```
CREATE TRIGGER PromotionOfficeDefault
   BEFORE UPDATE ON People
   FOR EACH ROW
   BEGIN
      IF (OLD.group = 'phd' AND
         NEW.group = 'vip' AND
         NEW.office IS NULL)
      THEN
         UPDATE People
         SET NEW.office = 'Nygaard-355';
      END IF;
```

- Access both the old and the new value of group to determine a promotion
- Complex condition on several attribute values in the same row
  - If someone is moved from phd to vip, but does not have an office assigned, assign them to Nygaard-355

ira@cs.au.dk

Example: AFTER INSERT

```
CREATE TRIGGER ZombieOffice
    AFTER INSERT ON People
    FOR EACH ROW
    BEGIN
        IF (NEW.office NOT IN (SELECT room FROM Rooms))
        THEN
            INSERT INTO Rooms(room,capacity)
            VALUES(NEW.office,4);
        END IF;
```

- Access the new value of office and compare it to values in another table, as well as insert into this other table
  - This is where we can express very powerful constraints
    - If someone is inserted into People but has an office value that does not exist in rooms, we can fix that by simply adding their office to the Rooms table with default capacity of 4

ira@cs.au.dk

Views

- A **view** is a virtual table / a named query
  - defined as a function of **base tables** or other views
    - E.g. views BusyDays, Vips on base tables People, Meetings

```
CREATE VIEW Vips AS
  SELECT * FROM People
  WHERE `group` = 'vip';

CREATE VIEW BusyDays AS
  SELECT name, date
  FROM People, Meetings
  WHERE People.userid = Meetings.owner;
```

https://dev.mysql.com/doc/refman/8.0/en/create-view.html

ira@cs.au.dk

Using views

- Since a **view** is a virtual table / a named query may use them just like any other table

```
SELECT *
  FROM BusyDays, Vips
  WHERE BusyDays.name =
        Vips.name;
```

```
CREATE VIEW Vips AS
  SELECT * FROM People
        WHERE `group` = `vip´;
CREATE VIEW BusyDays AS
  SELECT name, date
        FROM People, Meetings
        WHERE People.userid =
                Meetings.owner;
```

Corresponds to subquery that corresponds to the view definition

```
SELECT * FROM (
        SELECT name, date
                FROM People, Meetings
                WHERE People.userid = Meetings.owner) AS BusyDays ,
        (SELECT * FROM People
                WHERE `group` = 'vip') AS Vips
        WHERE BusyDays.name = Vips.name;
```


ira@cs.au.dk

Materialized Views

- Views may alternatively be stored as tables

```
CREATE MATERIALIZED VIEW VipsM AS
  SELECT * FROM People
  WHERE group = 'vip';
```

  - This view is actually stored like a table
    - requires recomputation whenever the view may have changed
  - Not available in MySQL – can often be simulated with triggers
    - Core idea: create a new table that is updated when the view should be recomputed

ira@cs.au.dk

Materialized Views trade-off

- Materialized view is (compared to regular view)
  - faster for queries
  - slower for modifications
    - Again, the reason is that queries can directly access the result of the query, which is already stored as a table, and does not have to be computed on the fly using the base table(s)
- Typically compromise
  - Recompute the view periodically
  - This only works if correctness is not critical
    - mailing advertisement to customers
    - computing pie charts from sales statistics

ira@cs.au.dk

Modifying Views

```
CREATE VIEW AvgCap AS
  SELECT AVG(capacity) AS average
  FROM Rooms;
UPDATE AvgCap SET average=117;
```

1. The average value is recomputed
2. The average value is stored in the view
3. The average value is stored in the base table
4. The average value cannot be updated
5. I don't know

ira@cs.au.dk

Modifying Views

- Generally, it makes no sense to update a view

```
CREATE VIEW AvgCap AS
  SELECT AVG(capacity) AS average
  FROM Rooms;
```

```
UPDATE AvgCap SET average=117;
```

- The function is not reversible...

ira@cs.au.dk

Modifiable Views

- Particularly simple views may be modifed
  - only a single table in FROM
  - only SELECT simple attributes
    - So, no aggregates or the like
  - no subqueries in WHERE


```
CREATE VIEW Vips AS
     SELECT * FROM People
     WHERE `group` = 'vip';

INSERT INTO Vips VALUES ('Glynn Winskel',
     'Turing-222', 'gwinskel', 'vip');
```


ira@cs.au.dk

Alternatives to Modifiable Views

- For more complex views on several tables or with aggregates, etc

```
CREATE VIEW BusyDays AS
  SELECT name, date
  FROM People, Meetings
  WHERE People.userid = Meetings.owner;
```

```
CREATE VIEW CountMe AS
  SELECT owner, COUNT(*)
  FROM Meetings
  GROUP BY owner;
```

- CountMe view not simple enough for modiable views (aggregate), but could potentially admit certain updates to base table: e.g. increment by creating a new meeting with default values
- BusyDays view not simple either (two base tables), but could admit some updates: e.g.if inserting name and date, create new meeting with default values otherwise, if the user exists
- Triggers may be used to catch view modifications
  - **INSTEAD-OF** Triggers
    - The intended action is then performed on the underlying base tables
    - This allows human insight to be used
    - BUT…. not available in MySQL

ira@cs.au.dk

Instead-of trigger example

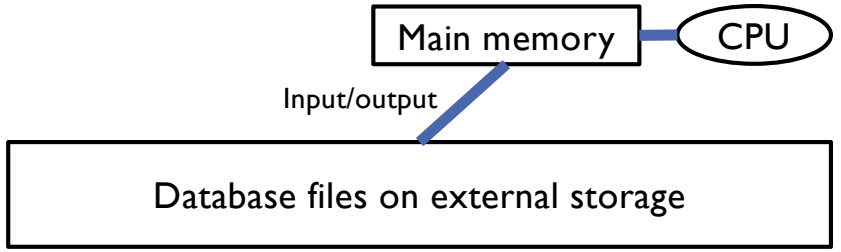
```
CREATE VIEW BusyDays AS
  SELECT name, date
  FROM People, Meetings
  WHERE People.userid = Meetings.owner;

CREATE TRIGGER BusyDaysDelete
  INSTEAD OF DELETE ON BusyDays
  REFERENCING OLD ROW AS OldDay
  FOR EACH ROW
  DELETE FROM Meetings
  WHERE date = OldDay.date AND
        owner IN (SELECT userid
                  FROM People
                  WHERE name = OldDay.name);
```

- BusyDays is join of People and Meetings projected to name and date
- Instead of deleting from the join, we could instead delete from the corresponding base table Meetings for a particular date and where we find the corresponding name in the People table

ira@cs.au.dk

Making data access faster

- Where is the data actually stored?
  - Usually, just a collection of files
  - Typically on external storage
    - Hard disks: provide relatively large storage at relatively low cost; non-volatile (considered permanent)
    - Recently, increasingly SSDs (flash storage), some of the same storage principles, but slightly different characteristics



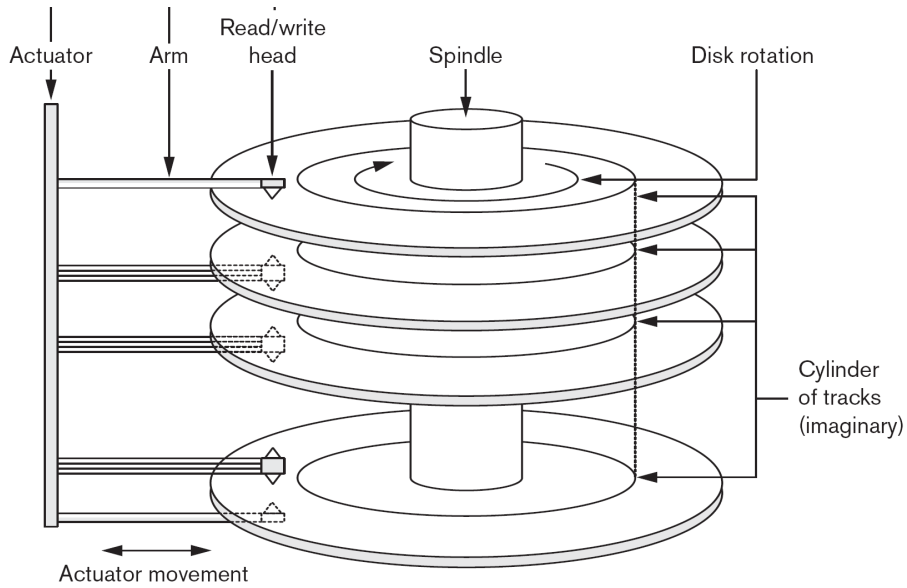
- Where can we actually work with the data?
  - CPU needs to load the data from external storage to main memory
    - RAM: relatively limited storage at relatively high cost; volatile (non-permanent)
- For the database to work with any of the stored data, need to load it to main memory first
- When done write any changes to external storage

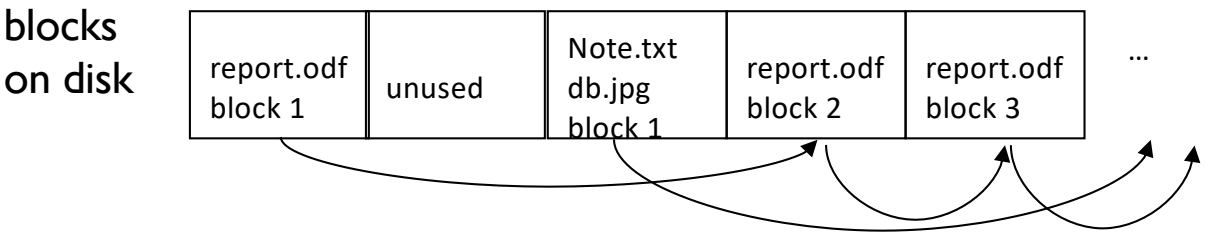
Note: there is more to the memory hierarchy, and main memory databases exist – out of scope for this course

ira@cs.au.dk

Disk Concepts – Files

- To read or write data from a disk, find the right location on the disk, then read from there
  - Seek time
  - Read time
- File systems: input/output (I/O) entire blocks (size OS dependent, e.g. 4KB)
  - Typically many records in each block, but large files may span several blocks
    - Blocks not necessarily contiguous (can be reorganized)



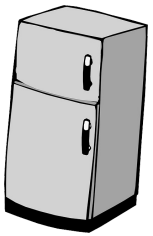


ira@cs.au.dk

I/O model for runtime

- Contiguous block I/O faster than random read (avoids disk seek)
- One disk access = 1,000,000 RAM accesses! (cache even faster)

- Justifies "count only I/Os" model of complexity
  - Means: "measure" response time as how many blocks you read from disk
  - Example: read 20 tuples from a relation
    - Slow: repeat 20 times: go find the record, read it into main memory
    - Better: go find the first record, read all 20 (if they are roughly in the same place = same block), transfer all of them to main memory in one go
  - Intuition: you sit on the couch, you need something to drink and eat
    - Slow: go to fridge, find soda, bring to sofa; go to fridge, find snack, bring to sofa
    - Better (in terms of time, not health): go to fridge, find both, bring to sofa
    - Number of trips is much reduced (= number of I/Os)
  - SSDs have no read/write heads and thus no seek delay; still substantially slower than RAM



ira@cs.au.dk

Selection

```
SELECT *
  FROM R
  WHERE condition;
```

- Full table scan
  - Read all rows in the table
  - Report those that satisfy the condition
  - Complexity in I/O model?

- Fine if many rows will actually be selected
  - Rule of thumb is 5-10%

ira@cs.au.dk

Point Query evaluated

```
SELECT *
  FROM People
  WHERE userid = 'amoeller';
```

- We know that `userid` is a key
  - **Point query**: looking for a single particular data point

- Optimization if `People` is sorted on `userid`
  - Full table scan can stop sooner

    Whenever we find the userid amoeller, we know we are done and do not need to keep going
    - great for amoeller, not so great for zenzen

ira@cs.au.dk

Binary search

```
SELECT * FROM People WHERE userid = 'amoeller';
```

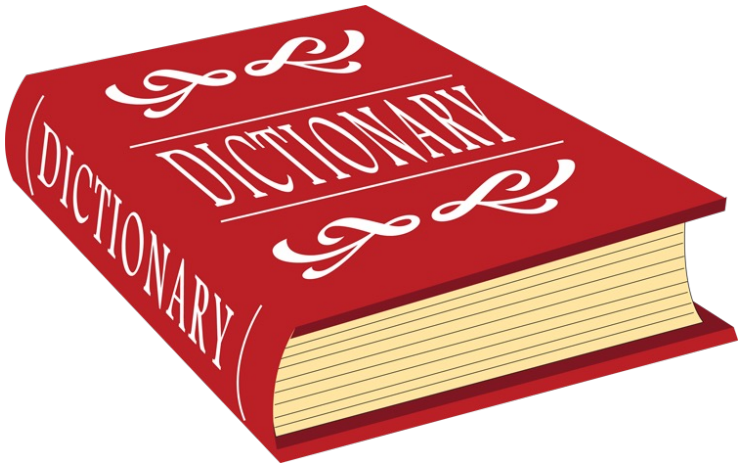
- Binary search not necessarily better
  - Jump to the middle of the list of userids
    - Check if current userid is before or after amoeller in alphabet
      - E.g. motte is after so jump to earlier point
        - Jump to first quarter, check again
          - E.g. froehr is after
            - Jump to first eight
              - E.g. aknonimos is before, so jump to later point
                - E.g. amoeller, check
                - Is found, done
  - Random disk access vs. sequential access
    - Full table scan is sequential access (go to fridge and bring as much as you can in one go), where binary search is random access (go to fridge, find an item quickly, make return trip for next item)
- So, what can help us?

ira@cs.au.dk

Indexes (= index structures)

- Index on a table: data structure that helps find rows quickly
- A table may have several indexes

```
CREATE INDEX DateIndex
  ON Meetings(date);
CREATE INDEX ExamIndex
  ON Exams(vip,date,time);
```



- Think of this as a virtual sorting of the table
- Each primary key has by default an index

- Pros and cons of indexes
  - Make (some) queries faster, make modifications slower

https://dev.mysql.com/doc/refman/5.6/en/create-index.html

ira@cs.au.dk

Which queries benefit from indexes?

```
CREATE INDEX ExamIndex ON Exams(vip,date,time);
```

A. Benefits all queries on Exams
B. Benefits queries using only vip, date, and time
C. Benefits queries such as a range query on date
D. Benefits only join queries on Exams

ira@cs.au.dk

Indexed File

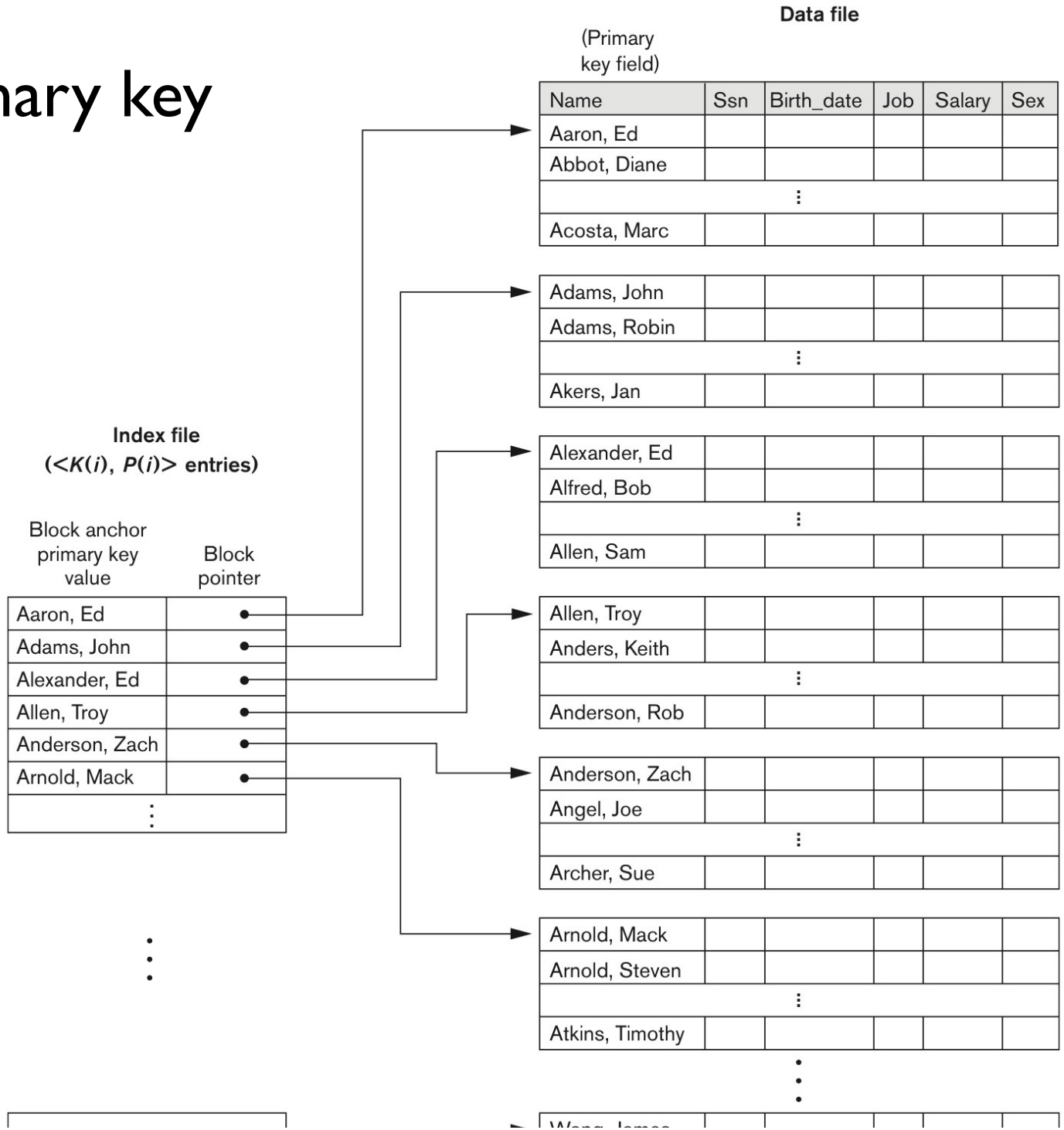
- Suitable for applications that require random access
- Usually combined with sequential file
- A **single-level index** is an auxiliary file of entries <search-key, pointer to record> ordered on the search key
- Index is separate from data file
  - Usually smaller

    10-20% rule of thumb, take with a grain of salt!
  - Can have multiple indexes on same relation

ira@cs.au.dk

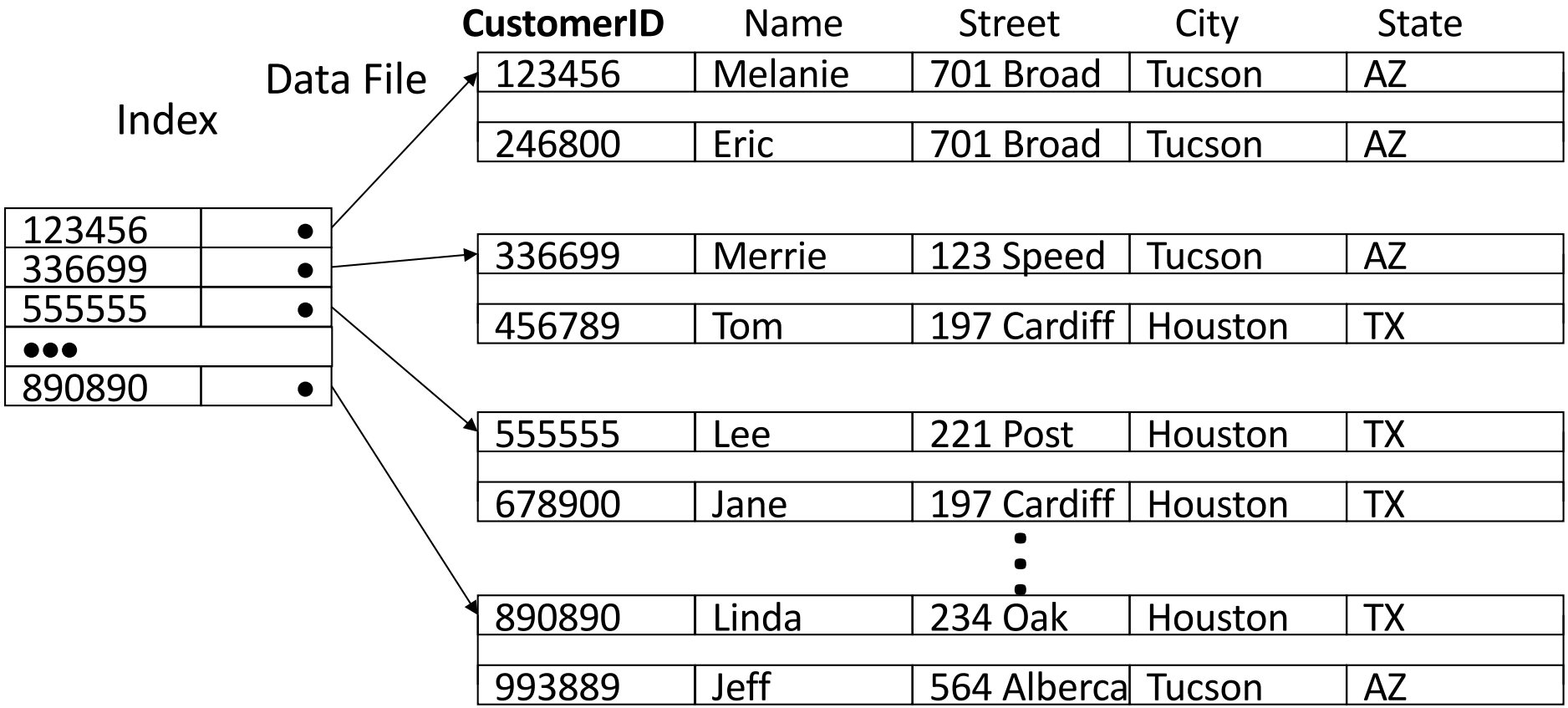
Index on primary key

- Often, index file will be in main memory
  - Means: look up primary key value "for free" as no I/O (no trip to fridge, key list placed on couch table already)
  - Once located, follow block pointers to disk block where the primary key value is found



Primary Index

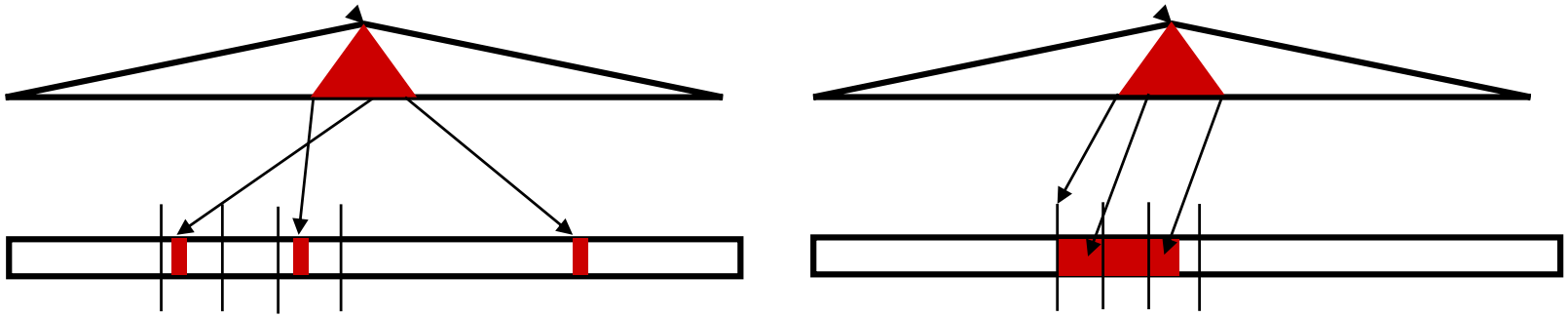
- **Primary Index**: defined on a data file ordered on the primary key
  - **Dense index** has one entry for each search key value
  - **Sparse index** - fewer index entries than search key values



ira@cs.au.dk

Clustering Index

- Generally, indexed rows are scattered in the table
- A **clustering index** has consecutive rows

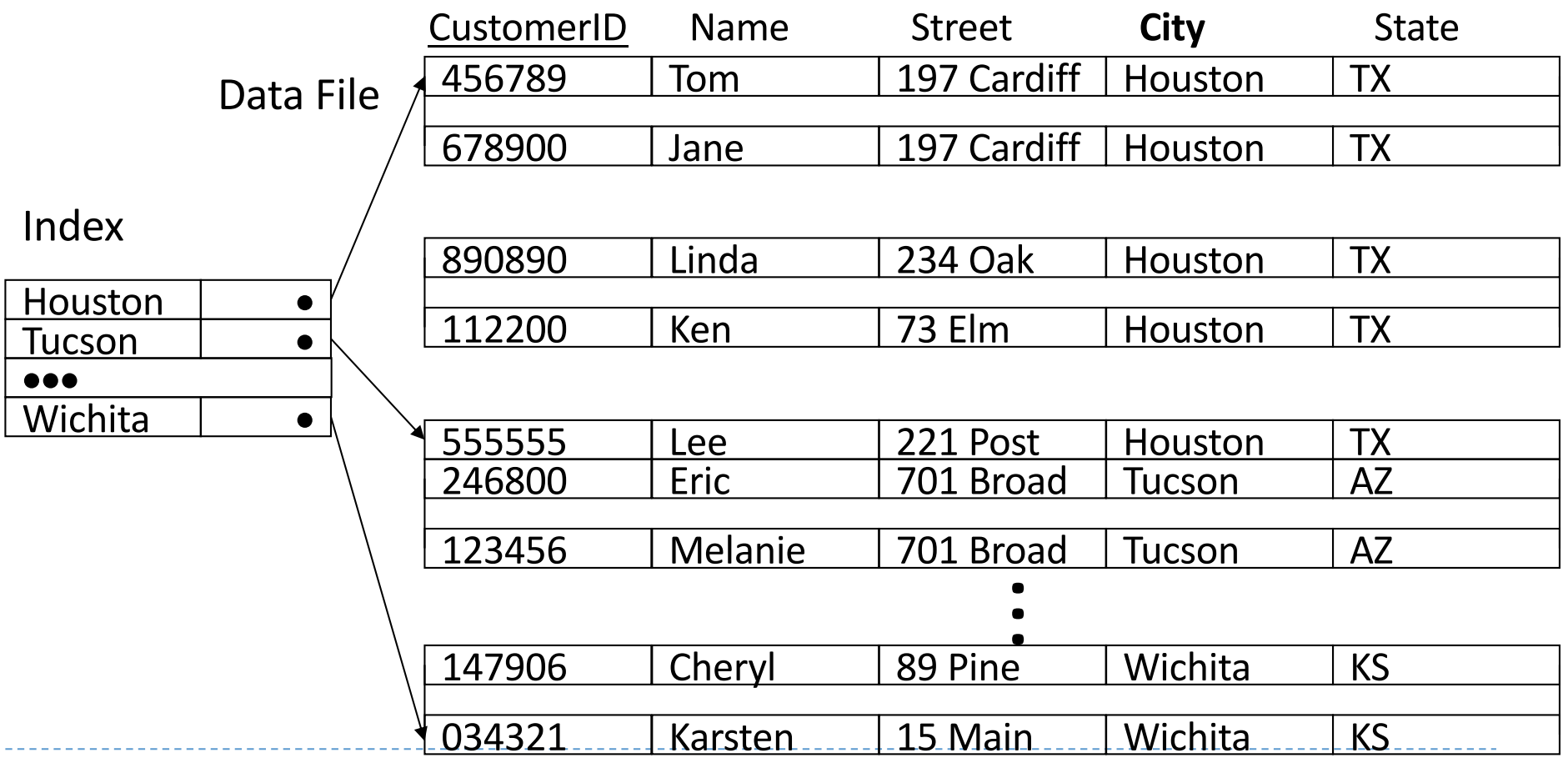


- Equivalent to sorting the table
- At most one index can be the clustering index
  - But other indexes may happen to be clustered too, attributes may be correlated

ira@cs.au.dk

Clustering (Dense) Index

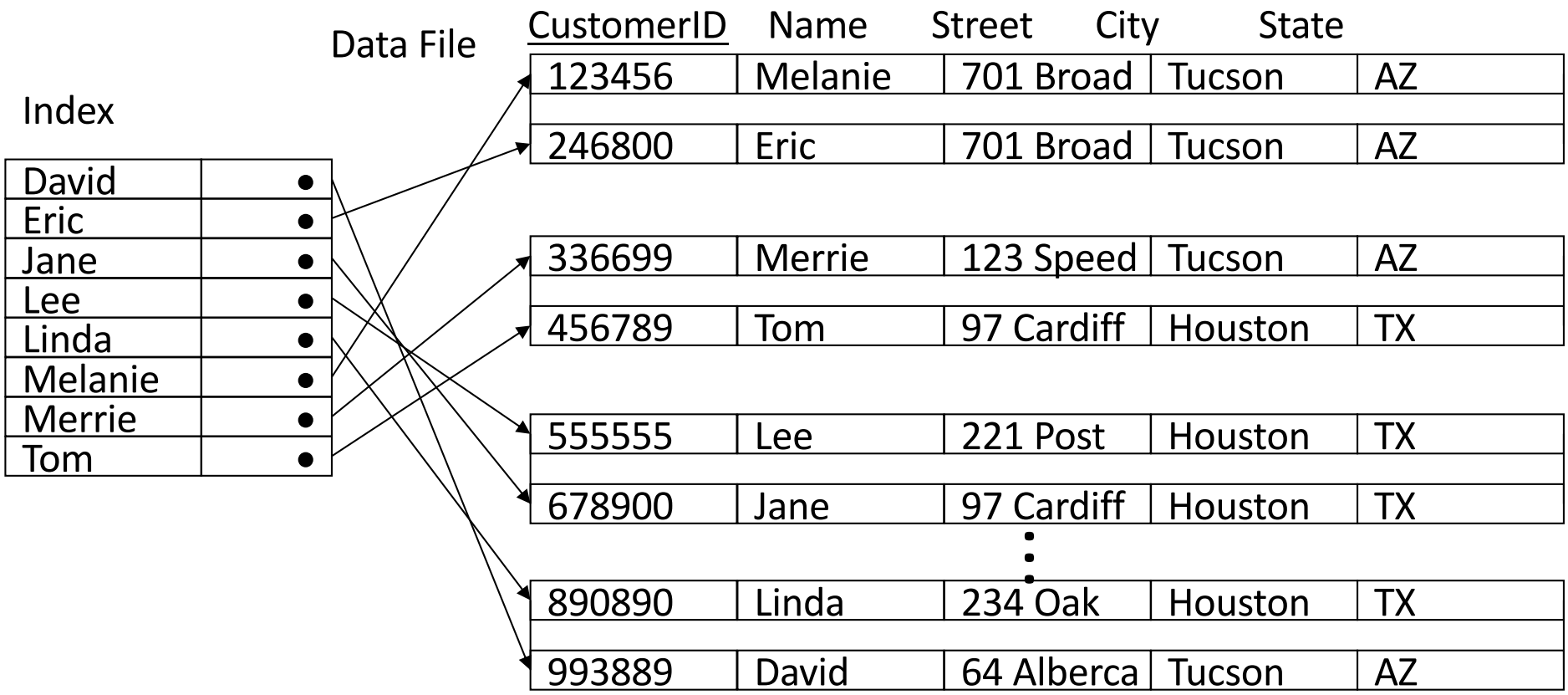
- **Clustering Index**: defined on a data file ordered on a non-key field
  - One index entry for each distinct value of the field, points to first data block of records for search key



ira@cs.au.dk

Secondary (Dense) Index

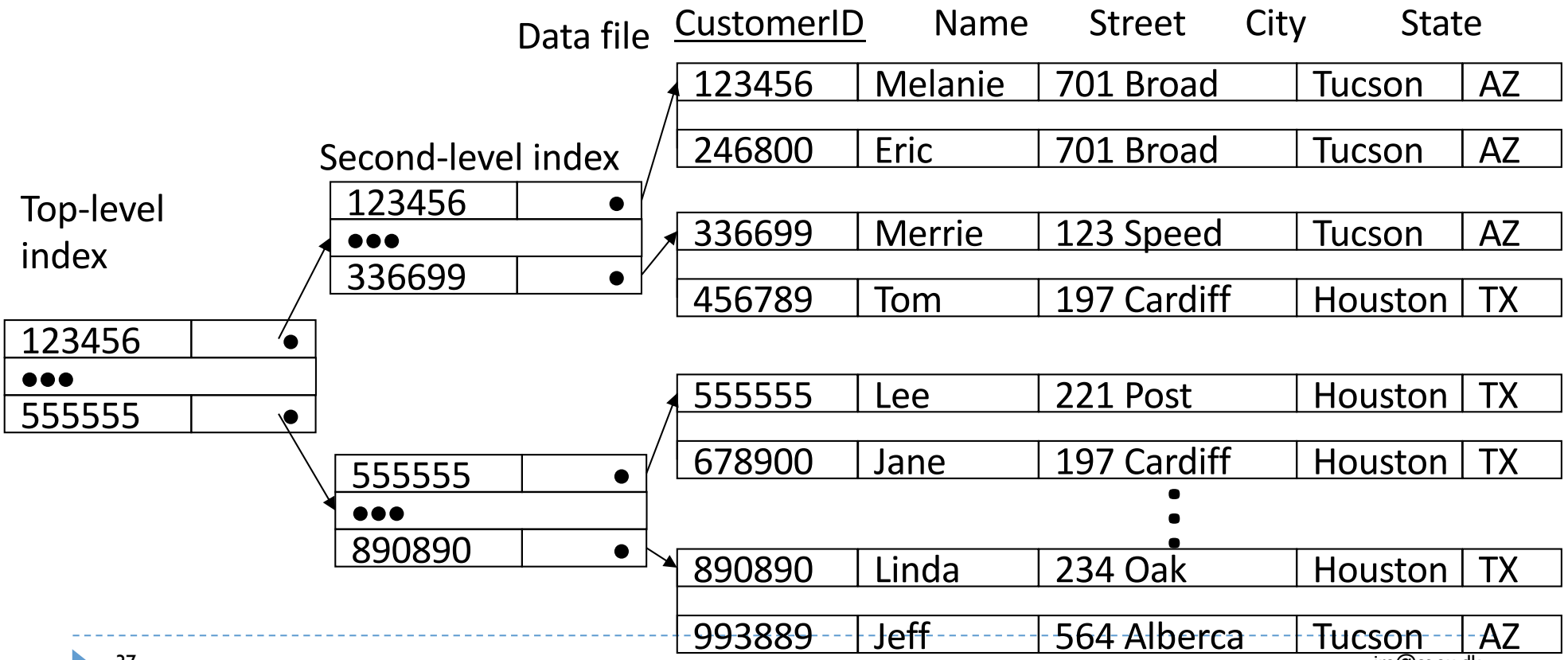
- **Secondary Index**: defined on a data file not ordered on the index's search key



ira@cs.au.dk

Multi-level Index Example

- **Multi-level index**: index on index, until all entries of the top level fit in one disk block
  - Every level of the index is an ordered file
  - Pin top-level index in main memory (RAM)



ira@cs.au.dk

Multilevel index

What is true for multilevel indexes (top level is the one in main memory)?

A. All levels of the index must be sparse
B. All levels of the index except for the bottom-most-level index must be sparse
C. All levels of the index must be dense
D. All levels of the index must be dense except for the bottom-most-level index

ira@cs.au.dk

Summary

- Be able to
  - Use triggers to define complex constraints
  - Use views as named queries
  - Discuss core principles in major indexing approaches to speed up database access

ira@cs.au.dk

Where to go from here?

- We know how to ensure complex constraints with triggers
- how to use indexes to speed up queries

- Next, we'll see how to use a database not as a standalone tool, but as part of a software program, as well as how to authorize access to (parts of) a database

ira@cs.au.dk

What was this all about?

Guidelines for your own review of today's session

- Triggers take the form…
  - They allow us to specify…
- Views are…
  - They are used as follows…
- Data is stored in files…
  - They are laid out as…
  - When considering runtime, the single most important aspect is…
    - E.g. the fridge example…
- Indexes are…
  - We use them in order to…
  - Sparse vs. dense means…
  - Clustered and multi-level indexes are…
  - Multiple indexes can be used by…

ira@cs.au.dk